68000ファミリ ハンドブック

68000·68008·68010·68020

W·クレイマー＋G·ケイン·著
岡本　茂·訳

68000

68008

68010

68020

啓学出版

「ハードウェアハンドブックとして、本書は、ハンディなレファレンスマニュアルを求めている、経験のあるマイクロプログラマやシステムデザイナにとって、もっとも便利である」と、米Byte誌、1986年9月号で紹介。

主要目次

Designed by Yellow Bag

68000ファミリ ハンドブック

68000・68008・68010・68020

W・クレイマー＋G・ケイン・著
岡本 茂・訳

68000

68008

68010

68020

啓学出版

訳者まえがき

本書は，W. Cramer and G. Kane ; 68000 Microprocessor Handbook ; Includes 68008, 68010 & 68020 第2版（Osborne/McGraw-Hill, 1986）を翻訳したもので，68000から68020に至る68Kファミリを解説したものである．訳者は先に68000の優秀さに注目し，それをザ68000（共立，1983）にまとめたが，68008，68010，68020についてもきちんとまとめたいと考えていた所へこの翻訳の話がきたので，早速原著を通読してみた．これは68Kファミリをマイクロプロセッサという立場からまとめ，アーキテクチャについて論じたもので訳者の考えと共通する所があったので，喜んでその翻訳を引受けることにした．

英語と日本語では文法が異なり，直訳したのでは硬い文章になってしまうので，原稿に何度も手を加え，読みやすくなるように心掛けた．必要な場合には脚註を入れ，分りやすくなるように心掛けた積りである．用語には十分注意したが，カナ書きのものも可成りある．コンピュータに関係した部分は進歩が速く，新語が次々と登場するので，止むを得ず訳者が作った新語も多少ある．これは基準が定まり次第に改めることとしたい．

ソフトウェア関係は原著には書いてないが，コプロセッサを含めた68020のアセンブラの解説はあった方がよいと考えたので，日立プロセスコンピュータエンジニアリング（株）の加藤木和夫氏にまとめていただいた．これはオペレーティング・システムやコンパイラなどでは必要である．

本書の出版に当り，多くの本を参考にし，多数の方々の御協力をいただいた．また啓学出版（株）の編集部の方々，特に鈴木信行氏には企画の段階から御世話になった．ここに記して厚く感謝する．

1987年春　　　　岡本　茂

序　　文

「68000 マイクロプロセッサ　ハンドブック」の初版は，Motorola 社の最初の 16 ビット・マイクロプロセッサ MC68000 についてまとめたものである．内部では 32 ビットでデータとアドレスを処理するというアーキテクチャを基本として，68000 がマイクロプロセッサを進歩させる長く続く道程の始まりに過ぎないことが，これによって明らかにされた．実際，Motorola 社は既に 68000 に続くいろいろなマイクロプロセッサを出している．

新しいマイクロプロセッサは最初の 68000 と命令上の上位互換性をもっているが[*1]，それらの間には大きな相違点かある．ソフトウェアという観点からみると，マイクロプロセッサが新しくなればアドレッシング・モードやデータ・タイプや命令も追加されて新しくなる．一方ハードウェアという観点からみると，68000 マイクロプロセッサ・ファミリに最近仲間入りしたもの[*2]はアドレス・バス幅，データ・バス幅，ピンの配置，信号，周辺とのインターフェース，消費電力などで，従前のものより進歩している．

こういう変化があったので，著者は初版の改訂版を出すべき時機がきたと決

＊1　68008 だけは別だが，他は 68000 よりも上位にある．

＊2　68000 から 68020 に至るもののうち 68020 を指す．

心した．我々は，それぞれの新しいマイクロプロセッサに関する詳細を，Motorola 社の援助によって得たことを感謝する．また，8 ビット，16 ビット，32 ビットなどのすぐれたマイクロプロセッサとその周辺素子という広大な範囲を含むように 68000 ファミリを拡張した Motorola 社に賛辞を捧げる．

William D. Cramer
Gerry Kane

目　次

日本語版補遺

第1章

序　　論

Motorola 社は 68K ファミリという一連のマイクロプロセッサによって，16/32ビット・アーキテクチャの世界に寄与した．68000 から始まって 68020 に至るこの流れにおいて，* Motorola 社はマイクロプロセッサ・テクノロジーにおける真のリーダとなることを目指し，首尾一貫した特徴のある製品を作ることを目的として，常に改良に励んでいる．

この 68K ファミリは，コンピュータを使う上での複雑さを少なくし，リアルタイムやマルチユーザにおける応用でも，高速で生産性の高い製品を作る基準を規定するものである．このため命令が上位互換性をもっており，もっと複雑な計算処理が必要になったとしても，下位のマイクロプロセッサで開発した応用プログラムを上位のプロセッサに移植できるという特徴がある．

68Kファミリ

ここでは，このファミリに含まれるものをさっと述べることにしよう。

＊ さらに 68030 が進行中である．1987 年 7 月にはサンプル出荷されよう．

68000　68000は16ビットの世界におけるMotorola社の最初の製品である．17個の32ビット・データ・レジスタとアドレス・レジスタ，14種類のアドレッシング・モード，16メガバイトのアドレス空間，56個の基本命令をもち，命令をパイプラインで処理し，5種類のデータ・タイプをサポートする．16ビットのデータ・バスと24ビットのアドレス・バスを特徴とする．

68008　このマイクロプロセッサは，68000の能力を基本としており，そのパッケージのコスト面で配慮が払われている．データ・バスは8ビットなのでボードのレイアウトも簡単になっており，バイト単位のメモリや周辺装置をもっと手軽にアクセスできるようになっている．

68010　このマイクロプロセッサは，仮想メモリと仮想マシンのハードウェア・サポート，多重ベクタ・テーブル，効率的なループ命令を備えている．

68012　このマイクロプロセッサは，30ビットのアドレス・バスを用いる点を除けば，68010と同じである．

68020　データ・バスとアドレス・バスが完全に32ビットで，4ギガバイトのアドレス空間をもち，(たとえば浮動小数点用) コプロセッサ・インターフェース，7種類のデータ・タイプ，18種類のアドレッシング・モード，効率的なオンチップ命令キャッシュを備えている．

本書の目的

本書は68Kファミリの動作モード，アドレッシング，データ・タイプに関する明確な情報を機能的にまとめたもので，ファミリのそれぞれのタイミングと信号を正確に述べている．

これはハードウェア向きの本なので，命令セットについては

Kane, Hawkings and Leventhal:68000 *Assembly Language Programming* (Osborne/McGraw-Hill, 1981)

を参照されたい.*

信号とタイミングの表現について

信号は**アクティブハイ**，**アクティブロー**，またはハイかローで**アクティブ**である．アクティブハイ信号はハイ状態で動作するように機能し，ロー状態では効果をもたない．アクティブロー信号はロー状態で動作するように機能し，ハイ状態では効果をもたない．2つのアクティブ状態をもつ信号は2つの違った

---

* 68020については本書補遺を見られたい．

形で動作するが，その形はハイかローかで定まる．この信号はインアクティブ状態をもたない．

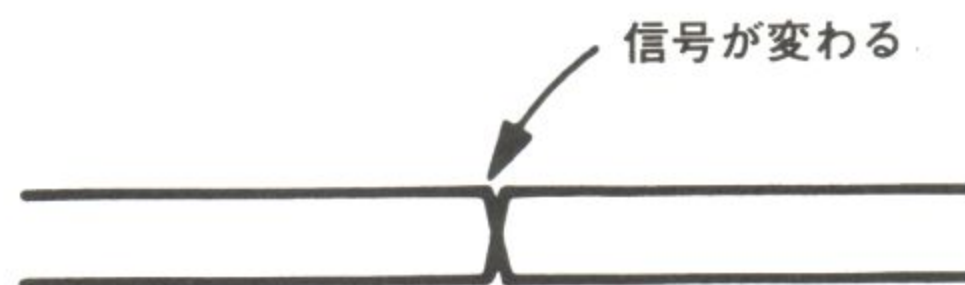
本書では，アクティブロー信号はその信号名の上に横線を引くこととする．たとえば $\overline{\mathrm{XX}}$ はロー状態でアクティブな信号である．アクティブハイ信号または2つのアクティブ状態をもつ信号は，YY のように信号名の上に横線を引かない．

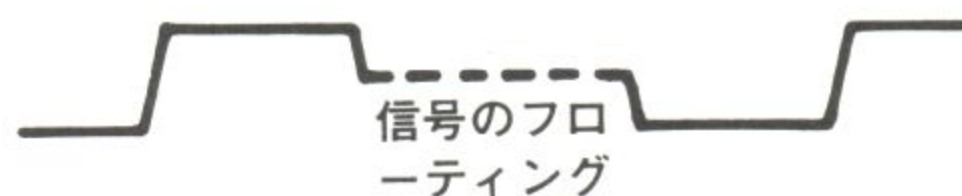
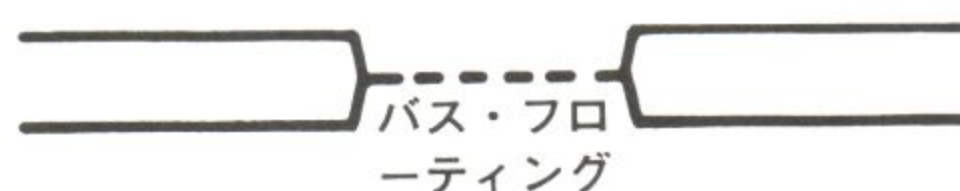
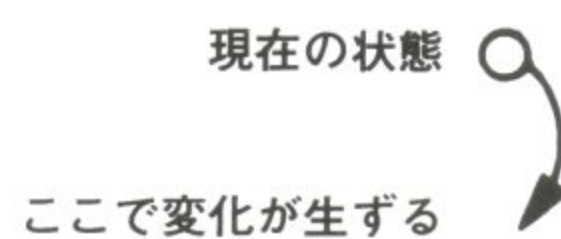
本文において,「**アサート**する」というのは，信号がインアクティブからアクティブに移ることを指し，ハイやローには関係しない．一方「**ネゲート**する」というのは，信号がアクティブからインアクティブに移ることである．実際の物理的な移行を図に示し，次のように規定する．

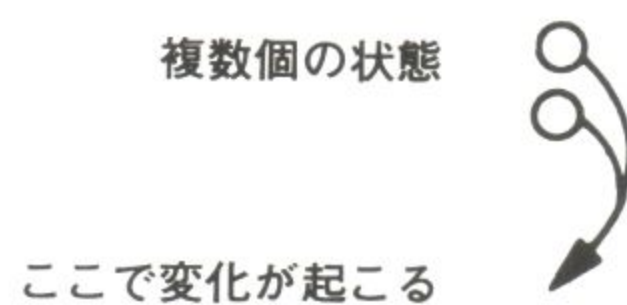
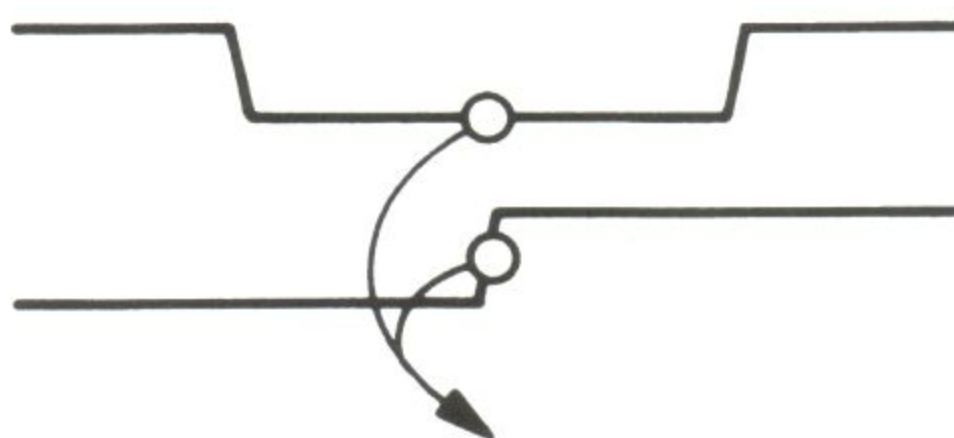
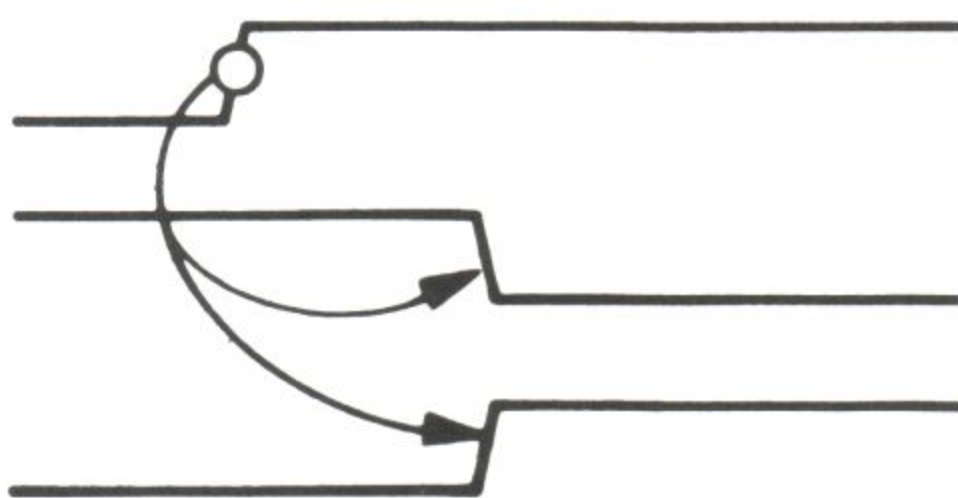
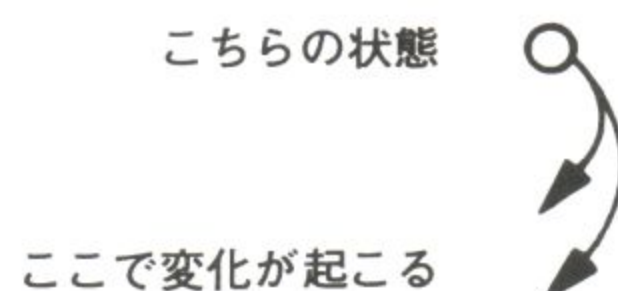
1．ロー信号が電圧をもたず，ハイ信号が電圧をもつ状態．

2．ローからハイに移行する信号を次のように示す．

3．ハイからローに移行する信号を次のように示す．

4．2つ以上の信号を並列で使うときは，

のように書く．これはそれぞれの信号のレベル変化を示し，移行時の状態(ハイからロー，またはローからハイ）を示すものではない．

5．3状態信号での**フローティング**（未定義）を次のように示す．

6．3状態バスでのフローティングを次のように示す．

7.　ある信号が別の信号を**トリガ**して変えるときは，その関係を矢印を使って次のように示す．

このように信号の変位が次の信号をトリガすることを次のように示す．

8.　2つ以上の状態によって別の動作がトリガされるとき，次のように書く．

したがって，ある信号がローからハイに移行し，ハイからローに移行する2番目の信号と結合して，3番目の信号での移行がトリガされることを，次のように表す．

9．　1個の移行で2つ以上の動作が生ずるとき，次のように書く．

したがって，信号がローからハイに移行して他の2つの信号で変化が生ずることを，次のように表す．

10．信号のレベル変化は矩形波で示し，立上り時間や立下り時間は無視する．

ファミリにおける名称について

68K ファミリにはいろいろなものが含まれており，その類似点は相異点よりも重要である．しかし新しいものは今までのモデルにない特徴をもっていると考えるのが自然であろう．そこですべてのものに共通な性質を述べるときには 68K という用語を使うこととし，そうでないときはそのモデル名によって区別することとした．たとえば「68000」は 68K ファミリの最初のものとして参照される．

本書の初版では，MC68000 というようにプロセッサ名に MC（Motorola Corporation を表す）という接頭辞をつけていたが，これは止めることとした．これは Motorola 社が 68K ファミリの各種プロセッサと機能の等しいチップの生産を多くの企業に対して認めているので，そういう企業を区別したくないからである．

第2章

機能の概観

この章では，68K ファミリの実行モード，レジスタ，利用できるアドレッシング・モードなどから，プログラミングの特徴を見る．

実行モード

スーパバイザ・ビット
ユーザ・モード
スーパバイザ・モード

68K は**ユーザ・モード**と**スーパバイザ・モード**という2つのモードのいずれかで動作し，その動作モードを定めるのは**ステータス・レジスタ**の中の1ビット*である．その名の示すように，プロセッサが応用レベルで動作するとき，ユーザ・モードになる．これに対し，スーパバイザ・モードは，オペレーティング・システムのレベルでのプログラムを目的としている．このモードはそのための**スタック・ポインタ**と**特権命令**をもっており，特権命令は，ユーザ・モードにおけるプログラムではやれないことも実行できるようになっている．

ファンクション・コード

プロセッサ・チップには，**ファンクション・コード**を出力するピン（FC 0-FC 2）があり，これは現在の実行モードとバスの状態（データ・アクセス，プ

---

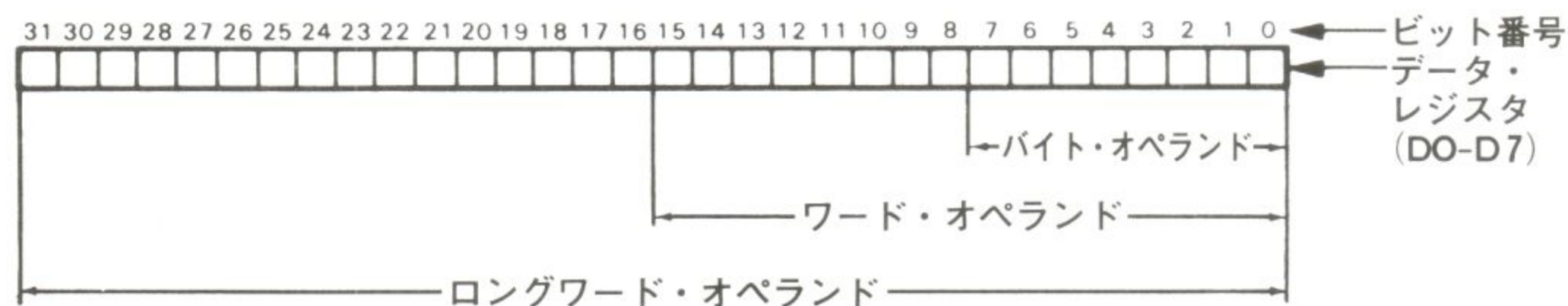
* Sビットという．これが0のときユーザ・モードである．

ログラム・アクセス，割込みアクノリッジ）を示している．ここに，実行モードとメモリの参照区分に関する情報があり，**メモリ管理ユニット**のような**外部ロジック**では，システム・メモリとユーザ・メモリに関してこの情報が使える．

ユーザ・モード・レジスタ

データ・レジスタとデータ・タイプ

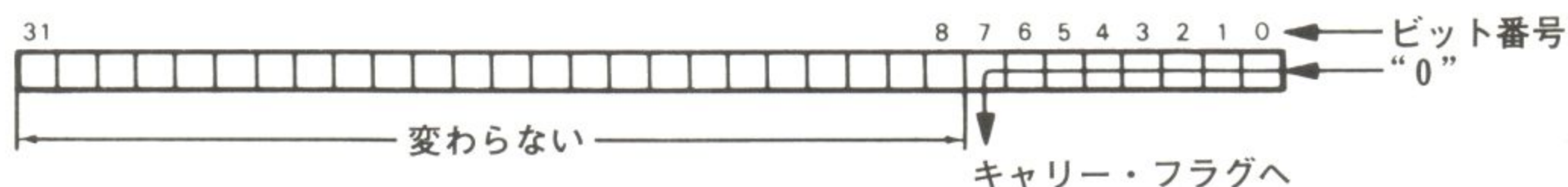
ユーザ・モードにおいて，68Kは8個の32ビット・**データ・レジスタ**（D0-D7），7個の32ビット・**アドレス・レジスタ**（A0-A6），32ビットの**スタック・ポインタ**（SP），32ビットの**プログラム・カウンタ**，8ビットの**コンディション・コード・レジスタ**[*1]をもっている．

データ・レジスタは，1ビットのデータ，8ビット（1バイト）のデータ，16ビット（1ワード）のデータ，32ビット（ロングワード[*2]）のデータを扱うことができ，その**最下位ビット**をビット0，**最上位ビット**をビット31として参照する．いろいろな大きさのオペランドをどのようにしてデータ・レジスタに設定するかを，次の図に示す．

1ビットに関する動作には，バイトやワードやロングワードにおける指定ビットのテスト，セット，クリアが含まれる．このようなビット操作命令では，直接またはデータ・レジスタを経由してビット番号を指定するのが普通である．

バイト・オペランドはビット0～7を占め，**ワード・オペランド**はビット0～15を占め，**ロングワード・オペランド**はビット0～31，すなわち，レジスタ全体を用いる．命令が**ソース・オペランド**または**デスティネーション・オペランド**としてデータ・レジスタを用いるときは，レジスタのそれぞれの部分だけが変わり，レジスタの上位ビットは変わらない．たとえばバイトを左に算術シフトする命令（ASL.B）[*3]では，次図のように，ビット0～7をシフトし，残りのビット8～31はそのままで変わらない．

＊1　図2.1を参照のこと．コンディション・コード・レジスタはステータス・レジスタの下位8ビットである．

＊2　基本の長さは1ワードで，ロングワードはいわゆる倍長ワードである．

＊3　アセンブラにおける記号

データ・レジスタを命令のソース・オペランドまたはデスティネーション・オペランドとして用いるほかに，**インデックス・レジスタ**や**ループ・カウンタ**として使うこともできる．

68020では「**クォードワード**」を扱うこともできる．クォードワードの長さは64ビットで，2つのロングワードの積と32ビットの除数で割る被除数をストアする目的で，32ビットの乗除算命令だけで用いられる．このため，任意のデータ・レジスタを2つまとめて使うが，どの2つを組み合わせてもよい．このとき，一方が上位（ビット32〜63），他方が下位（ビット0〜31）になる．

68020では，このほかに「**ビット・フィールド**」も扱える．すなわち，32個までの連続したビットを，バイト境界やワード境界やロングワード境界で限定せずにプログラムでアクセスすることができる．

アドレス・レジスタ

68Kには7個の汎用**アドレス・レジスタ**A0-A6がある．アドレッシング・モードによって，これらはオペランドのアドレス，オペランドに対するポインタのアドレス，ベース・アドレス，インデックスなどになる．アドレス・レジスタは16ビットまたは32ビットのデータをもてるが，アドレス・レジスタに対するバイト単位の直接のオペレーションはない．

ソース・オペランドとして直接に用いるときは，その直前に16ビットの値を符号拡張*1し（それより上位の部分は何の効果ももたず，命令による影響を受けない），デスティネーション・オペランドとして直接に用いるときは，16ビットのソースを符号拡張し，32ビット全体をデスティネーション・アドレス・レジスタとする．

スタック

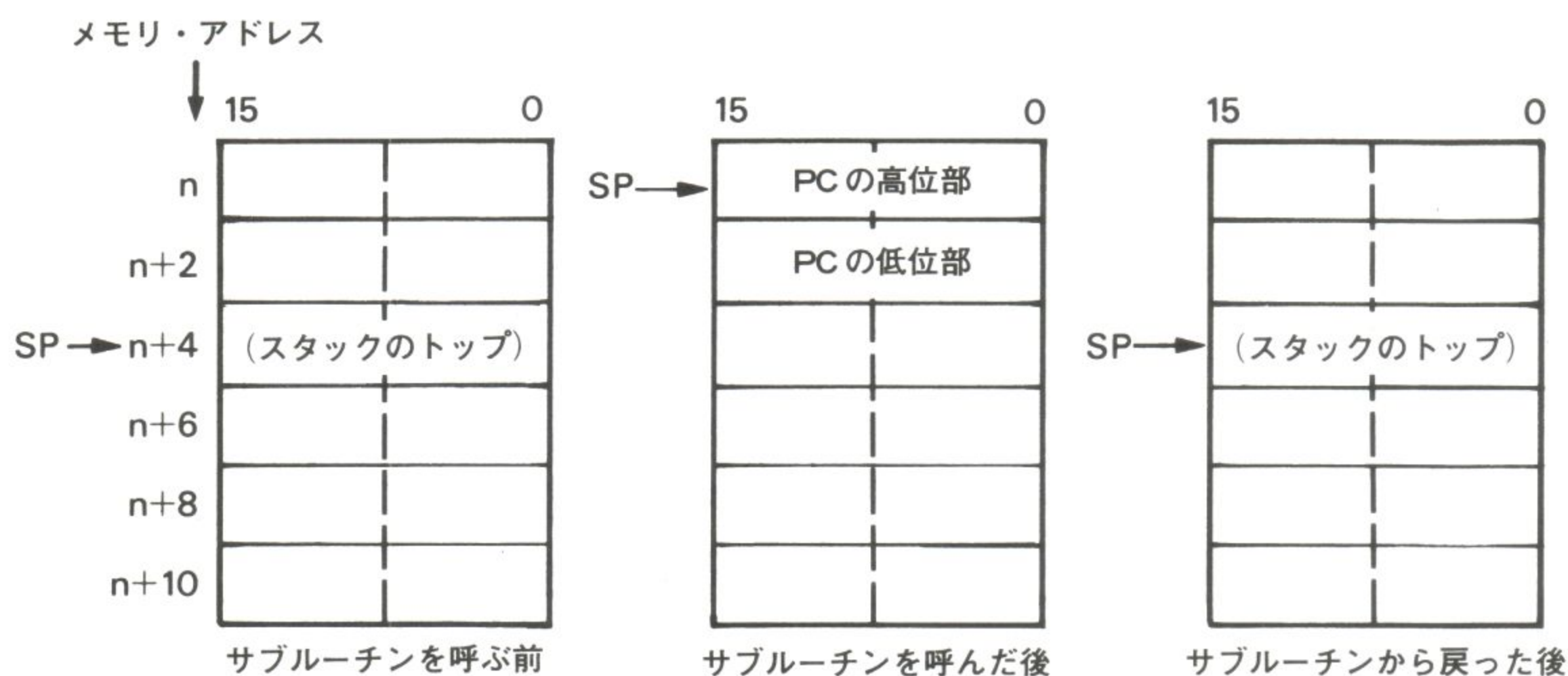
8番目のアドレス・レジスタA7は，**スタック・ポインタ**として用いられる．前述のように，現在のモードに従って，プロセッサには2つのスタック・ポインタA7とA7′があり，A7を**ユーザ・スタック・ポインタ**（**USP**），A7′を**システム・スタック・ポインタ**（**SSP**）という．どのスタック・ポインタを使っているかは，現在の実行モードによって知ることができる．なお後章に述べるように，68020には2種類のSSPがある．

スタックの**プッシュ***2と**プル***2を行うため，それぞれプリデクリメント/ポス

＊1　最初にある符号ビットの値をその前に置いて，たとえば16ビットによる値を32ビットで表現することをいう．$8123は$FFFF8123に，$4123は$00004123となる．

＊2　プッシュはスタックに押し込むことを，プルはスタックから引き出すことを指す．それに応じてスタックの高さが変わる．押込み引出しともいう．

トインクリメント間接アドレッシング・モードを用い，メモリ・アドレスの高い方から低い方へとスタックを満たしていく．たとえばサブルーチンを呼び出すと，スタック・ポインタは減ってプログラム・カウンタの低位部をプッシュし，もう一度スタック・ポインタが減ってプログラム・カウンタの高位部をプッシュする.*

サブルーチンから戻るときには，まずスタックからプログラム・カウンタの高位部をプルして，スタック・ポインタを増やし，その後，プログラム・カウンタの低位部をプルして，もう一度スタック・ポインタを増やす．これらの動作を次に図示する．

スタックはサブルーチンの呼出し処理や例外処理では必ず使われ，プログラム・カウンタの値をロングワードとしてプッシュ/プルするが，このときスタックは偶数（ワード）に合わせなければならない．バイトをプッシュ/プルするときは，ワードの上位が使われ，下位は使われない．これはスタックの境界が偶数であるという要求と一致する．

プログラム・カウンタ

プログラム・カウンタは特殊な32ビット・レジスタで，次に実行されるべき命令の中で先頭ワードの番地を示しているか，または命令をフェッチしている間，命令の次のワードの番地を示している．命令は偶数アドレス境界に合わせねばならず，プログラム・カウンタの内容は常に偶数である．

コンディション・コード

ユーザ・モードでは，ステータス・レジスタの下位バイト（**コンディション・コード・レジスタ，CCR**と略す）がアクセスされ，加減算やシフトなどのような命令の結果によって，各ビットの値が決まる．

「**キャリー(C)」ビット**は，加算による最上位ビットからの桁上げ(キャリー)や減算による最上位ビットの借り（ボロー）を示す．

---

* 16ビットのメモリを使っているので，32ビットのプログラム・カウンタをプッシュするには，2度に分けて行う必要がある．サブルーチンの戻りでも同じような注意が必要である．

「**オーバフロー（V）」ビット**は，算術演算後のオペランドの最上位ビットとその次のビットからのキャリーを排他的ORした値である．オーバフロー・ビットがセットされていることは，オーバフローが生じ，指定されたオペランドの長さでは結果が表せないことを示している．

「**ゼロ（Z）」ビット**は，演算の結果がゼロであるとセットされ，ゼロでない場合はクリアされる．

「**ネガティブ（N）」ビット**は，符号つき演算の結果の最上位ビットの値に等しい．これがセットされていると結果は負であり，クリアされていれば結果は正または0である．

「**拡張（X）」ビット**は，倍精度算術演算で用いられ，これが命令の影響を受けるときはキャリー・ビットに等しい．

ユーザ・モード・レジスタを**図2.1**に示す．

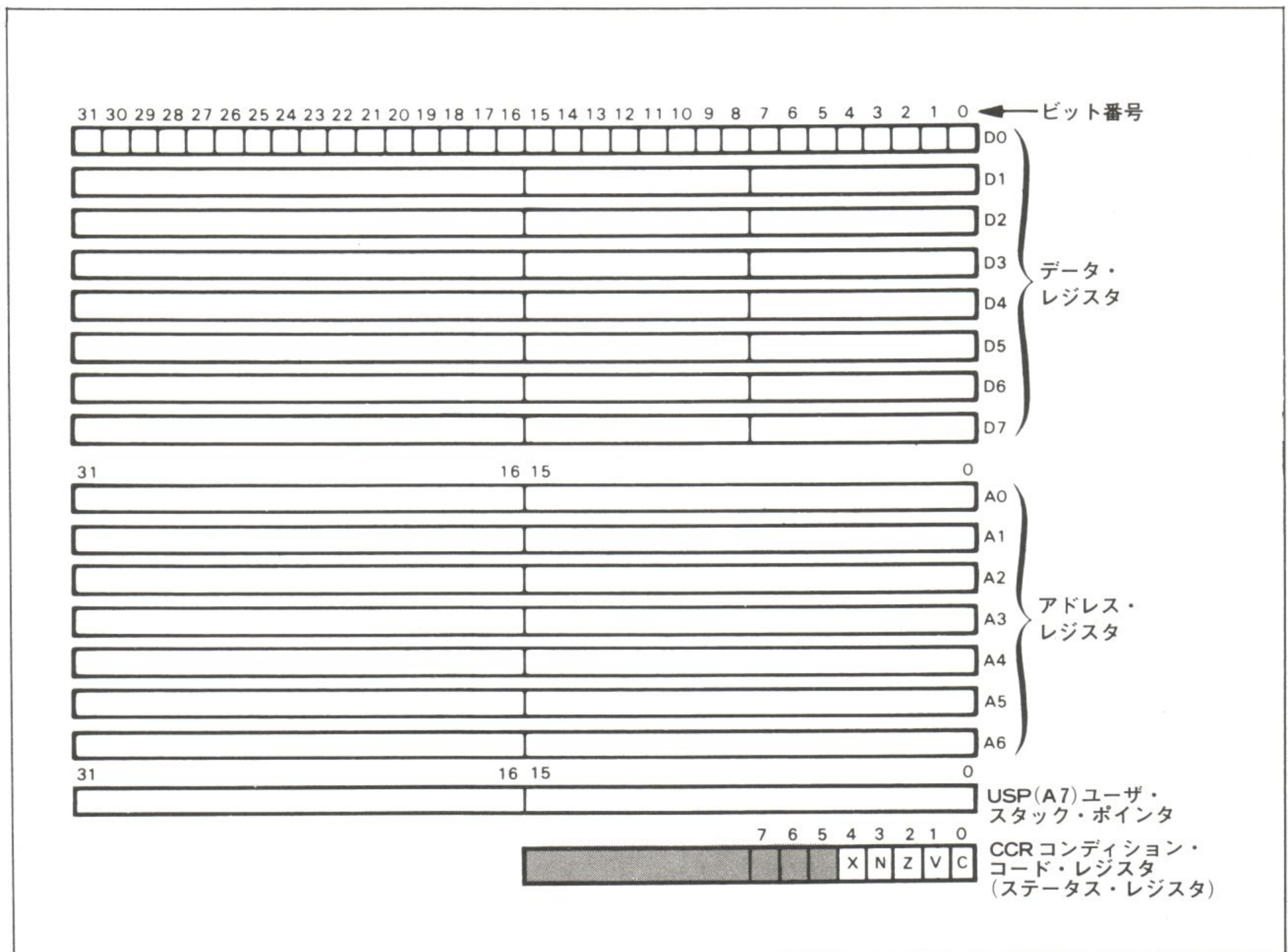



図2.1　ユーザ・モード・レジスタ

システム・モード・レジスタ

ユーザ・モードで利用できるデータ・レジスタ，アドレス・レジスタ，コンディション・コード・レジスタに加えて，68Kファミリのどのプロセッサもスーパバイザ・モード用の複数個のレジスタをもっている．この**スーパバイザ・レジスタ**の型や個数はそれぞれ異なるので，分けて述べることとする．特に断らぬ限り，下位モデルの特性は上位モデルにもあると考えてよい．

68000と68008

68000と68008は下位の製品であって，**図2.2**に示すように，2つのスーパバイザ・モード・レジスタをもっている．1つは**スーパバイザ・スタック・ポインタ**（A7′またはSSP）で，もう1つは**ステータス・レジスタの**上位バイトである．

スーパバイザ・スタック・ポインタは，ユーザ・スタック・ポインタのように32ビット幅で，メモリに下向きにプッシュする．スタックの参照は，(偶数番地から）整列したワードでなければならない．プロセッサがスーパバイザ・モードであれば，自動的にスーパバイザ・スタック・ポインタが使われる．このときユーザ・スタック・ポインタは使えない．*

ステータス・レジスタのシステム・バイトは，コンディション・コード・レジスタと結合して16ビットのレジスタになっており，2つの**フラグ・ビット**と3ビットの**割込みマスク**を含む．

「スーパバイザ（S)」ビットは，プロセッサの実行モードを特定するもので，セットしてあればスーパバイザ・モードであり，クリアしてあればユーザ・モードである．

「トレース（T)」ビットがセットしてあると，プロセッサは**トレース・モード**になり，命令を実行するたびに番号9のベクタを通して**トラップ**が発生する．

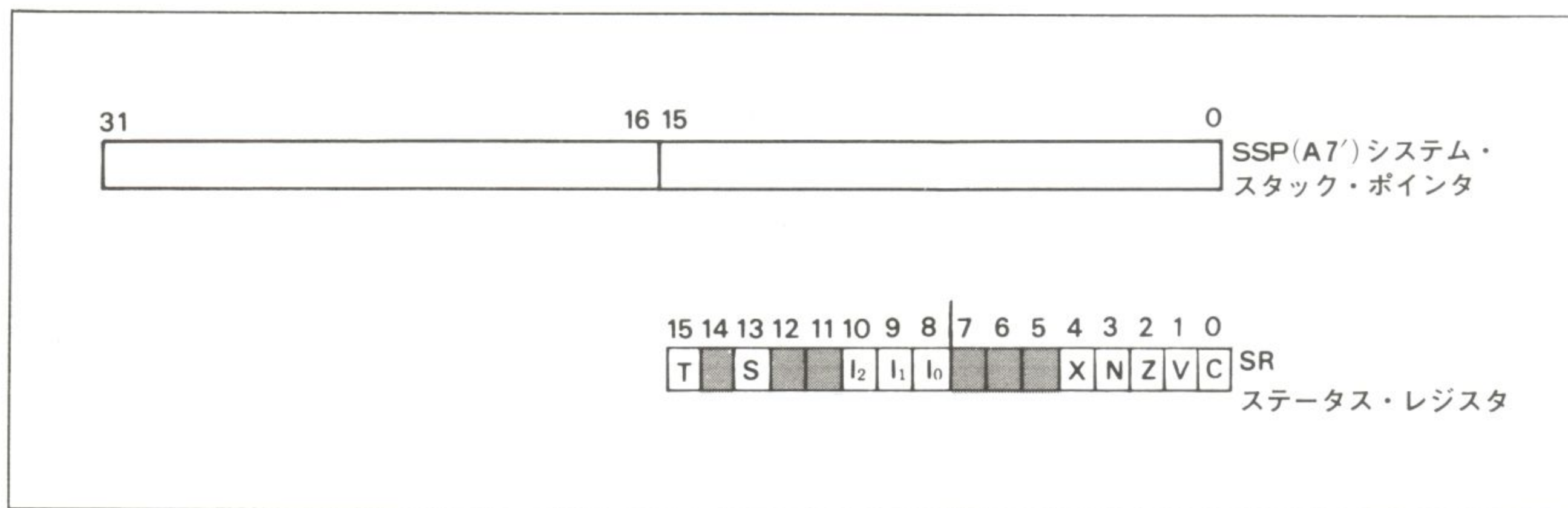



図2.2　スーパバイザ・モードで利用できるレジスタ（68000，68008）

＊　USPは別だからその中味は変わっていない．

このような**例外処理**については，第7章で述べる．トレース・モードでは命令を**シングル・ステップ・モード**で実行することがわかれば現在は十分である．これによって応用プログラムを1命令ごとに調べる**デバッグ用プログラム**が作れることになる．

68K プロセッサには，8レベルの**優先度**があり，そのどれでも動作する．外部デバイスは割込み要求ライン IPL 0 -IPL 2 を任意に組み合わせて信号をアサートし，プロセッサに割込みをかけることができる．この割込みレベルを表す2進数での値が，割込みをかけるデバイスの重要性を示している．システム・クロックのような高優先度のデバイスは，端末ドライバのような低優先度のデバイスよりも割込みレベルが高い．最高は7（2進で111），最低は0（2進で000）である．

ステータス・レジスタの**割込みマスク**は，プロセッサの現在の動作レベルを決める．普通のユーザ・モードではレベル0と決められており，デバイス・ドライバがもっと高いレベルで動作する．デバイスがプロセッサに割込みをかけたとき，このデバイスがIPL 0 -IPL 2 にアサートしたレベルを，割込みマスクにある値と**内部ロジック**によって比較する．

もし**割込み要求**が高レベルの優先度をもっていれば，プロセッサは例外処理を始める．そうでなければ，プロセッサ自身の優先度（割込みマスクにおける値）が下がるまで割込み要求を一時的に無視する．

この割込みマスクは，割込みラインと結合して非常に有効な優先度による実行の方法を規定しており，遅い周辺機器が普通のプログラムの実行に割り込めることやタイム・センシティブ・デバイス（時間を読出しできるデバイス）が遅い周辺機器に順に割り込めることが確実になっている．

例外処理については，ここに述べた以外にも多くのことがある．第7章において，例外処理を行う際のプロセッサと周辺機器の動作をまとめて論ずる．

68010 と 68012

68010 と 68012 には**スーパバイザ・スタック・ポインタ**と**ステータス・レジスタ**のシステム・バイトに加えて，32ビットの「**ベクタ・ベース・レジスタ**」と2つの**オルタネート・ファンクション・コード・レジスタ**（SFC と DFC）がある（次ページの**図2.3**）．

68000 と 68008 にはともに固定した例外ベクタ・テーブルがある．このテーブルは0番地から始まり，システム・レベルのソフトウェアにより，例外処理ルーチンのアドレスをロードするのに使われる．プロセッサが例外状態になると，これらのアドレスを通して自動的にそちらに進む．例外処理の詳細は第7章で論じるが，現在は 68000 と 68008 のベクタ・テーブルが0番地から始まっているというハードウェア上の事実と，68010 の場合はベクタ・ベース・レジスタ（VBR）によってベクタ・テーブルのスタート・アドレスを定めるので，

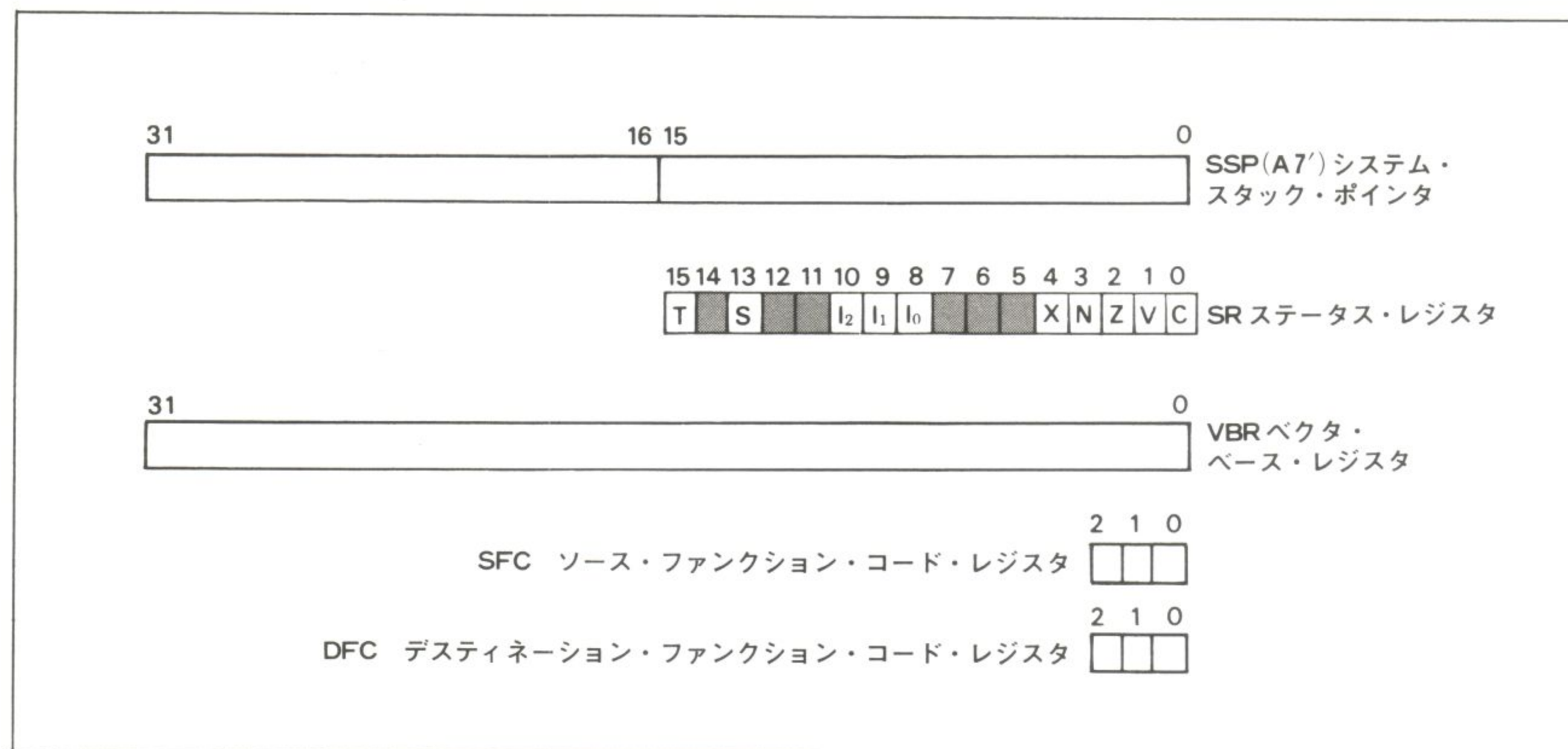



図2.3　スーパバイザ・モードで利用できるレジスタ（68010，68012）

そのためのソフトウェアを要するということだけを知っておけばよい．

すでに述べたように，68Kのチップ上には3つのファンクション・コードを出力するラインがあり，これらはプロセッサで現在進行しているアクセスの型をどんな周辺機器にも示している．標準としては，これらのモードがユーザ・プログラムのアクセス，ユーザ・データのアクセス，スーパバイザ・プログラムのアクセス，スーパバイザ・データのアクセス，割込みアクノリッジを示している（45ページの表5.2を参照）．

68010の**オルタネート・ファンクション・コード・レジスタ**（SFC，DFC）により，システム・レベルのプログラムはそれぞれのファンクション・コード出力を確定でき，（ステータス・レジスタに関する移動命令）MOVESでソースとデスティネーションをフェッチ/ストアしている間にこれが出力される．

68020

68020は他の68Kプロセッサにおいて論じられたスーパバイザ・レジスタをすべて用い，さらに2つの**オンチップ命令キャッシュ・レジスタ**をもち，またステータス・レジスタのスーパバイザ・バイトで2ビットを新たに用いている．これらを**図2.4**に示す．

キャッシュ・レジスタにより，オンチップ命令キャッシュがソフトウェアで行える．この命令キャッシュを使うと命令が非常に速くアクセスでき，ループになっている部分がこの中に含まれているとさらに高速で実行される．**キャッシュ制御レジスタ**（**CACR**）により命令キャッシュの制御とステータスがアクセスされ，**キャッシュ・アドレス・レジスタ**（**CAAR**）がキャッシュ制御ファンクションで必要なアドレスを保持する．

ステータス・レジスタの**トレース・ビット**は2つあってＴ０，Ｔ１という．新しいものはＴ０で，Ｔ１は従来68Kで使われていたＴである．この２ビッ

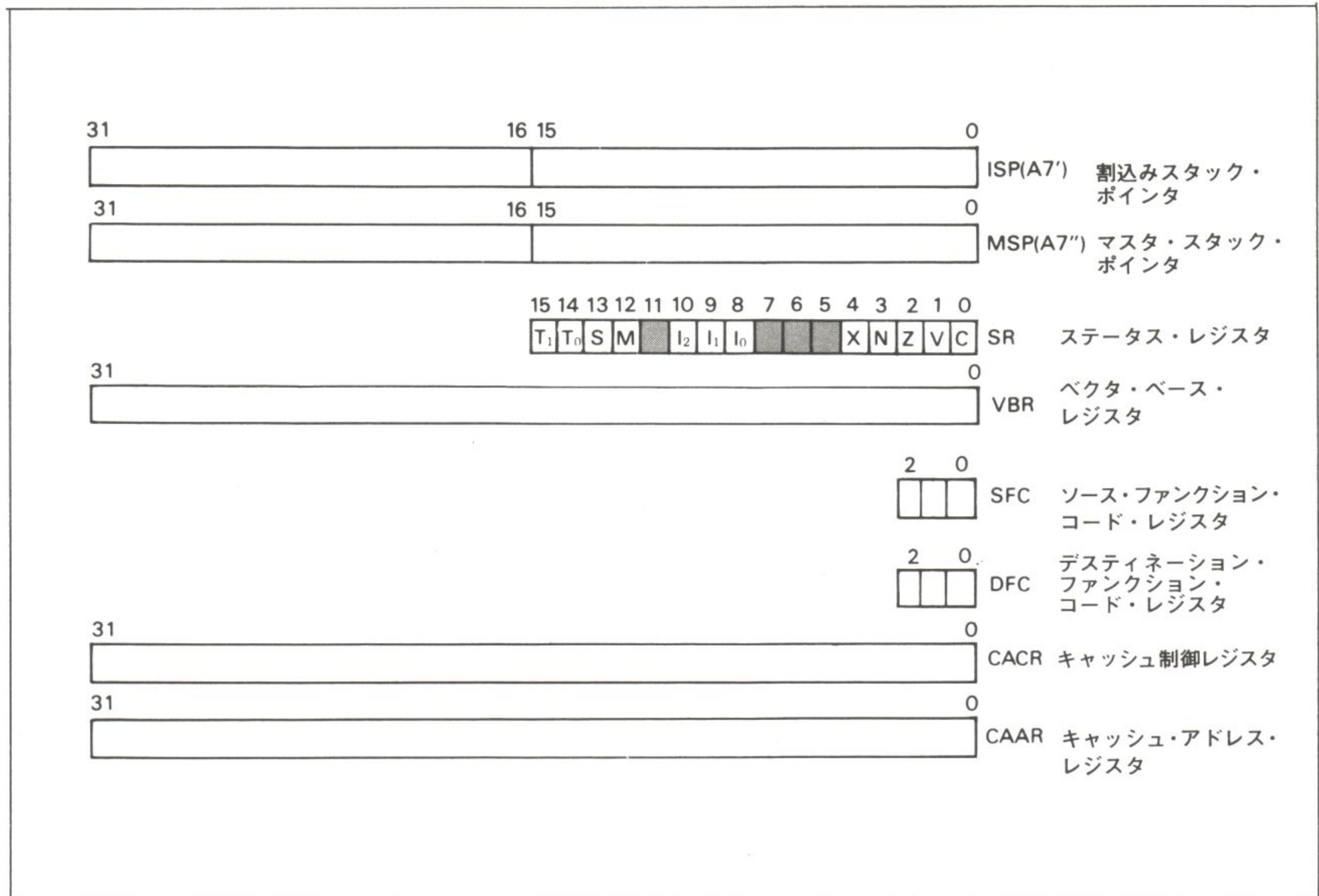



図2.4　スーパバイザ・モードで利用できるレジスタ（68020）

トにより，新しいトレース機能が追加された．これが00のときはトレースは行われない．01のとき，プログラムの流れが変わる所（ブランチやサブルーチンの呼出し）でのみトレース・トラップが立つ．10のときは 他の68Kのように各命令ごとにトラップが立つ．11は定義されていない．

68020ではスーパバイザ・スタックを**割込みスタック・ポインタ（ISP）**，**マスタ・スタック・ポインタ（MSP）**に分け，負担を二分した．Sビットがセットされているとき，ステータス・レジスタの**マスタ・ビット（M）**がどちらを使うかを定める．Mビットがクリアされていればプロセッサは割込みスタック・ポインタを使うが，これは普通の68Kプロセッサと同じである．もしMビットがセットされていると，プロセッサはマスタ・スタック・ポインタを使う．

これによって，入出力のような非同期例外とオペレーティング・システムのユーザ・タスクからの呼出しを，オペレーティング・システムで区別して扱えるので，2つのスーパバイザ・スタック・ポインタはマルチ・タスク・システムでは有効である．ユーザ・タスクはトラップ（TRAP）命令による実行サービスを要求できるから，インターフェースが「もっときれい」になったことを意味する．マスタ・スタック・ポインタはユーザからのリクエストに応えるためにタスク関連情報と一時的なストレージを保持し，割込みスタック・ポインタは一般に実行中のタスクとは独立な非同期例外処理で直接に用いる．

メモリ構成

バイトによる順序づけ

68K ファミリはメモリを**バイト，ワード，ロングワード**と分けてアクセスできる．アドレスはバイトの位置で定め，データが複数バイトを占める場合にはその先頭バイトの位置で定める．ワードの最下位バイトのアドレスはワードのアドレス＋1であり，ロングワードの最下位バイトのアドレスはロングワードのアドレス＋3である．バイト，ワード，ロングワードのメモリにおける**整列状態**を**図2.5**に示す．

アドレスづけ

68000，68008，68010，68012 では，ワードとロングワード（と命令語）は偶数アドレスから始まる必要がある．これはデータ・バスの構造に由来するもので，もしワードやロングワードを奇数アドレスでアクセスしようとすると，プロセッサは第7章に述べるような例外処理を始める．

68020 では奇数アドレスのワードのアクセスを調べ，この制限を除いている．もしそのようなワードが見つかると，そこで部分的に2～3回アクセスする．したがって実行上でオーバヘッドが生ずるので，できればワードをビット0がゼロに等しい偶数境界に並べ，またロングワードをビット0と1がゼロに等しいロングワード境界に並べる方がよい．

実行効率を最大にするには 68020 でもデータをワード境界から始める方がよく，* これは 68K ファミリに共通している．68020 ではシステム・スタックにロングワードをプッシュし，プルすることもできる．最もうまく動作させるためには，ロングワード境界でスタックを位置づけるべきであろう．

仮想メモリと仮想マシン

68K ファミリのプロセッサは多くのメモリをアクセスできる．しかし多くの例においては，実際に使える最大メモリをもつことは必ずしも経済的でない．68010，68012，68020 のどれも「仮想メモリ」という技術をプログラムで用いることができるが，これはシステムが計算機としてもち得る以上の十分大きなアドレス空間をもっているかのように思わせる技術である．

仮想メモリ

システムが物理的にはもっていないメモリをアクセスしようとすれば，メモリ管理ユニットはバスエラー例外ベクタでトラップを立て，「ページ・フォールト」を知らせてくる．そこでオペレーティング・システムは誤りの原因をその点で調べ，ディスクのような補助記憶装置上にあるメモリをアクセスしようとしたことが原因であれば，オペレーティング・システムは物理的なメモリにディス

＊　命令は偶数バイトから始まることになっている．

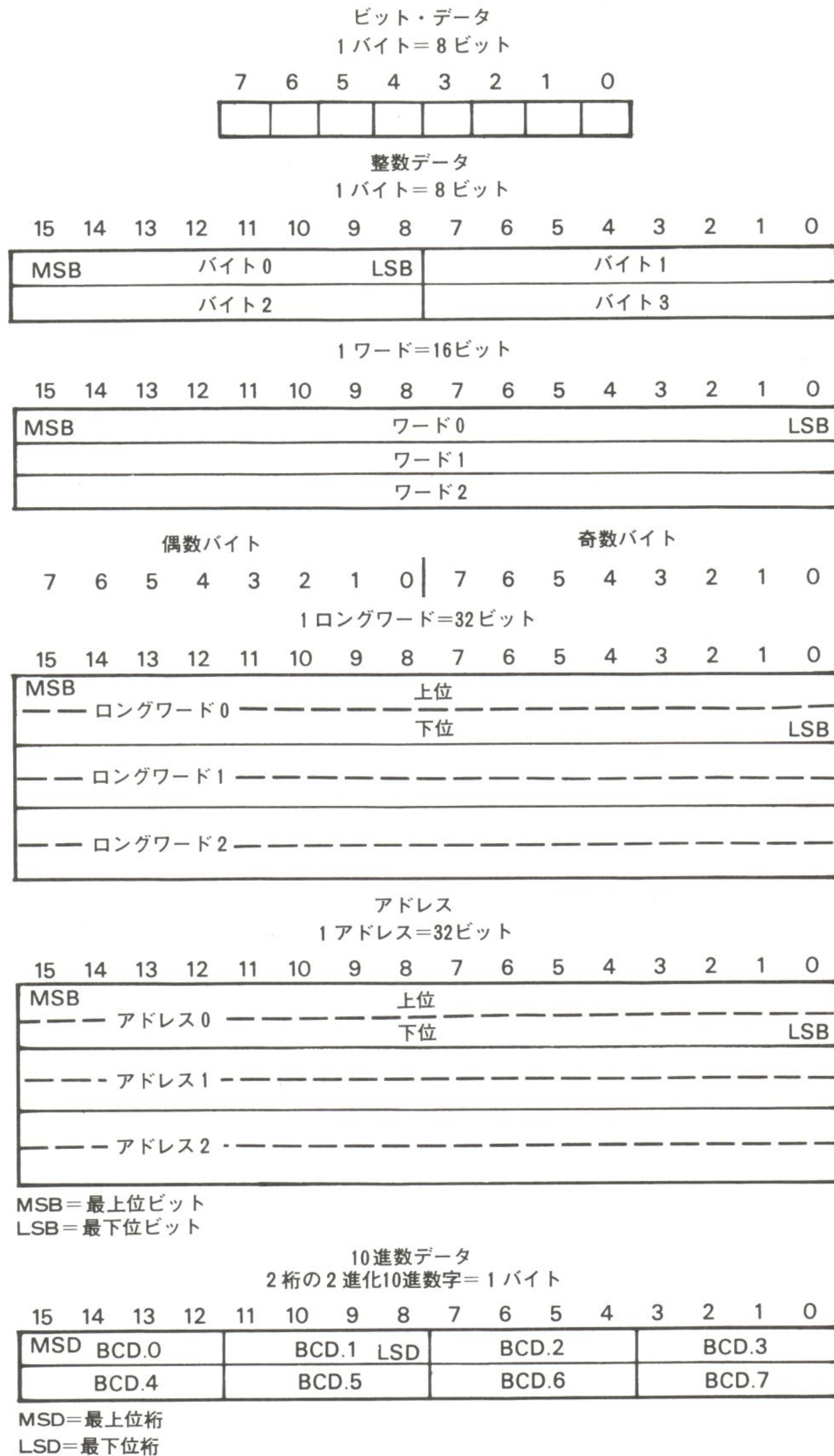



図2.5　メモリにおけるデータ構成

クの必要ブロックを読み出す.

こうして必要なデータがメモリにあれば，オペレーティング・システムが制御をユーザ・プログラムに渡し，プロセッサは「**命令継続**」を用いて，割込み命令を完了する．この方法を用いれば，プロセッサは多くの場合，命令の実行中かも知れないフォールト・バス・サイクルでプログラムの実行を続けられる．命令継続を適当に行うため，プロセッサは例外命令の開始に先立って，プロセッサの内部情報をセーブしなければならない．68010，68012，68020ではこれをスタックにセーブし，例外からの復帰命令（RTE；return from exception instruction）を実行してプロセッサにこの情報を戻す.

仮想マシン

システム開発環境にあっては，たとえばシリアル通信ポートのようなシステムに実在しないデバイスをアクセスするプログラムを書かねばならないことがある．これは浮動小数点コプロセッサに関する命令のようなもので，プロセッサにない命令を実行したいとき，「仮想マシン」ならば未定義のメモリ・アドレス（誤ったデバイスのステータス・レジスタなど）をアクセスするための例外処理を利用できる．また命令セットにない命令を実行するように，プロセッサに教えるときも同様である．このときソフトウェアによる例外処理で，このようなデバイスや命令をエミュレートできる.

第3章

アドレッシング・モード

68K ファミリには18種類の**アドレッシング・モード**があり，7つの基本的なタイプに分類されている．それらを次に示す.*

1. レジスタ直接アドレッシング
   a. データ・レジスタ直接
   b. アドレス・レジスタ直接
2. レジスタ間接
   a. アドレス・レジスタ間接
   b. ポストインクリメントつきアドレス・レジスタ間接
   c. プリデクリメントつきアドレス・レジスタ間接
   d. ディスプレースメントつきアドレス・レジスタ間接
   e. インデックスおよび8ビットまたはベース・ディスプレースメントつきアドレス・レジスタ間接†
3. メモリ間接
   a. ポストインデックスつきメモリ間接†

---

* 実は2のeを2種類として数えている.

b.　プリインデックスつきメモリ間接†

4.　プログラム・カウンタ間接

a.　16ビットディスプレースメントつきPC間接

b.　インデックスおよび8ビットディスプレースメントつきPC間接†

c.　インデックスおよび16-または32-ビットディスプレースメントつきPC間接†

5.　プログラム・カウンタ・メモリ間接

a.　ポストインデックスつきPCメモリ間接†

b.　プリインデックスつきPCメモリ間接†

6.　絶対アドレッシング

a.　絶対ショート

b.　絶対ロング

7.　イミディエイト

†のついたモードは68020で拡張された機能かあるいは68020のみで使われているものである．詳細についてはこの章の後で論ずるので，それを読んでいただきたい．

アドレス・エンコード

68Kの命令は固有の方式でオペランド・アドレスを**エンコード**する．最初のワードが命令を指定するが，オペランドのアドレスを含む場合も多い．それに続くワードがオペランドを指定している．この「**延長」ワード**は68000から68012まででは4種類あるが，68020ではさらに増えて10種類にのぼる．これは，アドレッシング・モードの追加によるものである．

実効アドレスは**モード**と**レジスタ**を指定するフィールドによってエンコードされる．このフィールドはそれぞれ3ビットあり，命令語の最初のワードに含まれている．それに続く延長ワードは絶対アドレッシングではオペランド・アドレスであり，メモリ間接モードでは大変複雑なアドレッシング・フィールドを含んでいる．この複雑な延長フィールドがインデックスづけや間接指定およびディスプレースメントの表現やその大きさを示している．

表3.1には命令語と延長ワードに関するアドレッシング・モードのフィールドがまとめてあり，22ページの**表3.2**にはモード/レジスタの値とその定め方がまとめてある．また，22ページの**表3.3**にはインデックス抑制およびインデックスと間接の選び方がまとめてある．モード/レジスタの組み方およびインデックス・ビットと間接ビットの意味は，次に述べるアドレッシング・モードの説明で明らかになろう．

表3.1　アドレッシング・モード・フィールド

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
OPコード										モード			レジスタ		

単一の実効アドレス形式

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
D/A	インデックス・レジスタ			W/L	スケール		0	ディスプレースメント							

延長ワードの簡易形式

D/A	インデックス・レジスタ	W/L	スケール	1	BS	IS	BDの大きさ	0	I/IS
ベース・ディスプレースメント（0，1，2ワード）									
外部ディスプレースメント（0，1，2ワード）									

延長ワードの最長形式（68020のみ）

レジスタ	データ・レジスタまたはアドレス・レジスタ（表3.2参照）
モード	アドレッシング・モード（表3.2参照）
OPコード	命令と可能なモード/レジスタ 次のオペランドに対する情報
ディスプレースメント	符号つき8ビット値
スケール	インデックスのスケール因子（68020のみ） 00＝1倍 01＝2倍 10＝4倍 11＝8倍

I/IS	インデックス/間接の選択（68020のみ，表3.3参照）
BDの大きさ	ベース・ディスプレースメントの大きさ（68020のみ） 00　保留 01　ディスプレースメントなし 10　ワード・ディスプレースメント 11　ロングワード・ディスプレースメント
IS	インデックス抑制（68020のみ，表3.3参照）
BS	ベース抑制（68020のみ） 0　評価後ベース・レジスタ加算 1　ベース・レジスタ抑制
W/L	インデックス・レジスタの大きさ 0　符号拡張ワード 1　符号つきロングワード
インデックス・レジスタ	データ・レジスタまたはアドレス・レジスタ（000－111）
D/A	インデックス・レジスタの型 0　データ・レジスタ 1　アドレス・レジスタ

表3.2　モード/レジスタ・エンコード

モード	レジスタ	アドレッシング操作
000	reg #	データ・レジスタ直接
001	reg #	アドレス・レジスタ直接
010	reg #	アドレス・レジスタ間接
011	reg #	ポストインクリメント・アドレス・レジスタ間接
100	reg #	プリデクリメント・アドレス・レジスタ間接
101	reg #	ディスプレースメントつきアドレス・レジスタ間接
110	reg #	インデックス・アドレス・レジスタ・メモリ間接*
111	000	絶対ショート
111	001	絶対ロング
111	010	ディスプレースメントつきプログラム・カウンタ間接
111	011	ディスプレースメントつきプログラム・カウンタ・メモリ間接*
111	100	イミディエイト・データ
111	101-111	予約済み
		＊68020のみ

表3.3　アドレッシング・モードのエンコード

IS	I/IS	アドレッシング操作
0	000	メモリ間接なしインデックス
0	001	外部ディスプレースメントのないプリインデックス間接
0	010	外部ディスプレースメント・ワードつきプリインデックス間接
0	011	外部ディスプレースメント・ロングワードつきプリインデックス間接
0	100	予約済み
0	101	外部ディスプレースメントのないポストインデックス間接
0	110	外部ディスプレースメント・ワードつきポストインデックス間接
0	111	外部ディスプレースメント・ロングワードつきポストインデックス間接
1	000	インデックスもメモリ間接もないもの
1	001	インデックスも外部ディスプレースメントもないメモリ間接
1	010	インデックスのない外部ディスプレースメント・ワードつきメモリ間接
1	011	インデックスのない外部ディスプレースメント・ロングワードつきメモリ間接
1	100-111	予約済み

アドレッシング・モード

データ・レジスタ直接

データ・レジスタ直接モードでは，データ・レジスタの内容がオペランドになる．

データ・レジスタ：　［オペランド］

アセンブラ記号：Dn
（実効アドレス：Dn）

アドレス・レジスタ直接

アドレス・レジスタ直接モードでは，アドレス・レジスタの内容がオペランドになる．

アドレス・レジスタ：　［オペランド］

アセンブラ記号：An
（実効アドレス：An）

アドレス・レジスタ間接

アドレス・レジスタ間接モードでは，アドレス・レジスタの内容がオペランドのアドレスになる（すなわちオペランドを「指示」している）．

アドレス・レジスタ：　［メモリ・アドレス］
メモリ：　［オペランド］
指定する

アセンブラ記号：An
（実効アドレス：(An)）

ポストインクリメントつきアドレス・レジスタ間接

ポストインクリメントつきアドレス・レジスタ間接モードでは，単純なアドレス・レジスタ間接と同じく，アドレス・レジスタの内容がオペランドのアドレスになる．ただし，このオペランドを用いた後で，アドレス・レジスタの内容にオペランドの長さが加えられる．すなわちバイトなら1，ワードなら2，ロングワードなら4をアドレス・レジスタの内容に加える．

注）アドレス・レジスタがスタック・ポインタでバイト単位の動作を行うときは，レジスタに2が加えられる．これはスタックの効率を大きくするためワードとして整列するからである．

アドレス・レジスタ：　［メモリ・アドレス］
オペランドの大きさ（1，2，4）：　(+)
メモリ：　［オペランド］
指定する

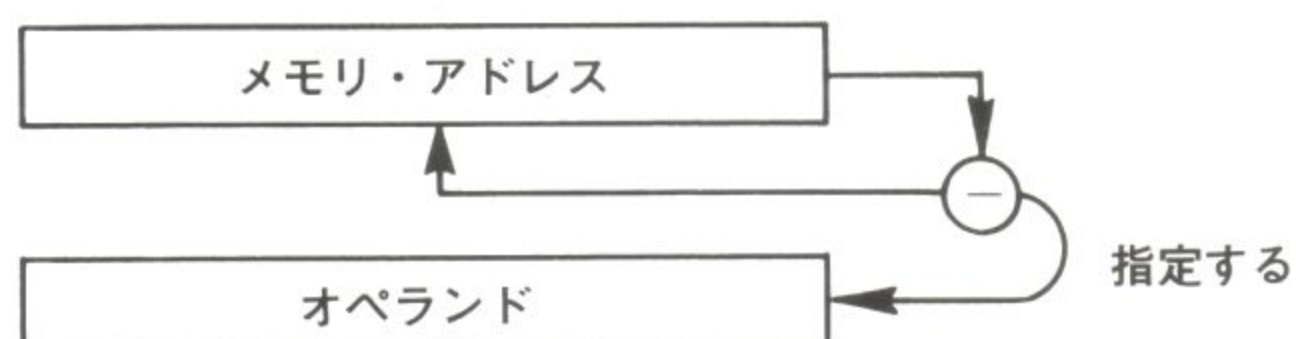
アセンブラ記号：(An)＋
（実効アドレス：(An)，An←An＋大きさ）

プリデクリメントつきアドレス・レジスタ間接

プリデクリメントつきアドレス・レジスタ間接モードでは，単純なアドレス・レジスタ間接と同じく，アドレス・レジスタの内容がオペランドのアドレスになる．ただしオペランドを用いるに先立ち，アドレス・レジスタの内容からオペランドの長さが引かれる．すなわちバイトなら1，ワードなら2，ロングワードなら4をアドレス・レジスタから引く．

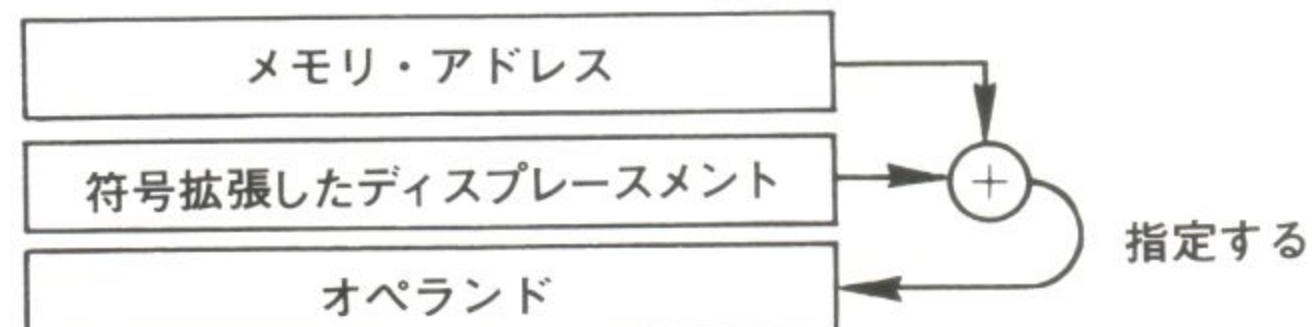
注）アドレス・レジスタがスタック・ポインタでバイト単位の動作を行うときは，レジスタから2が引かれる．これはスタックの効率を大きくするためワードとして整列するからである．

アドレス・レジスタ：　メモリ・アドレス
オペランドの大きさ（1，2，4）：　⊖
メモリ：　オペランド　　指定する

アセンブラ記号：－(An)
(実効アドレス：An←An－大きさ，(An))

ディスプレースメントつきアドレス・レジスタ間接

ディスプレースメントつきアドレス・レジスタ間接モードでは，オペランドの内容はアドレス・レジスタの内容と16ビット・ディスプレースメントの値を加えたものになる．ディスプレースメントは負にもなれるように32ビットに符号つきで拡張する．

アドレス・レジスタ：　メモリ・アドレス
16ビット・ディスプレースメント：　符号拡張したディスプレースメント　⊕
メモリ：　オペランド　　指定する

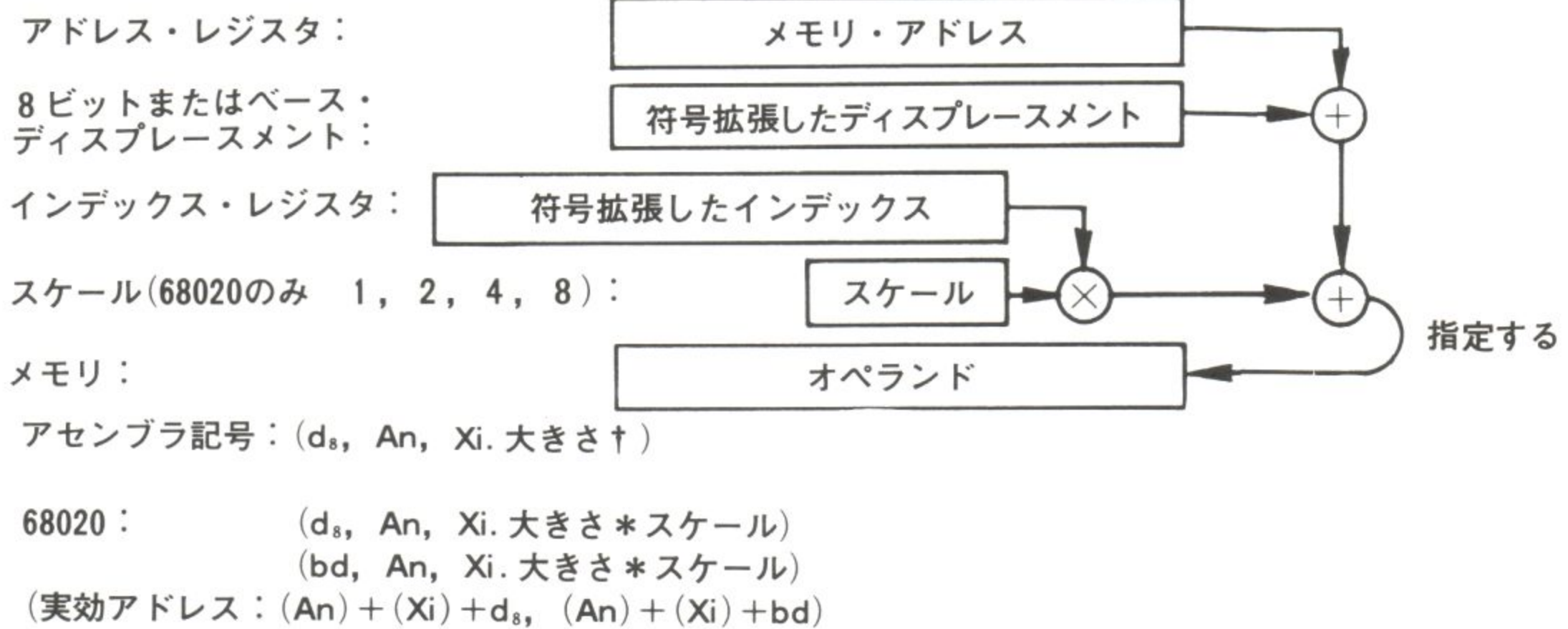
アセンブラ記号：(d_{16}，An)
(実効アドレス：(An)＋d_{16})

インデックスおよびディスプレースメントつきアドレス・レジスタ間接（8ビットディスプレースメントまたはベース・ディスプレースメント）

インデックスおよびディスプレースメントつき間接モードでは，オペランドの内容はアドレス・レジスタの内容とディスプレースメントおよびインデックス・レジスタにスケール因子を乗じたものの和になっている．

ベース・ディスプレースメントが8ビットか16ビットならば，それを32ビットに符号つきで拡張する．インデックス・レジスタ*の値は16ビットか32ビットだが，16ビットのときそれを32ビットに符号つきで拡張してから用いる．スケール因子は2^0(＝1)，2^1(＝2)，2^2(＝4)，2^3(＝8)のいずれかで，この**スケール**のついたインデックスをサポートしているのは68020だけである．

---

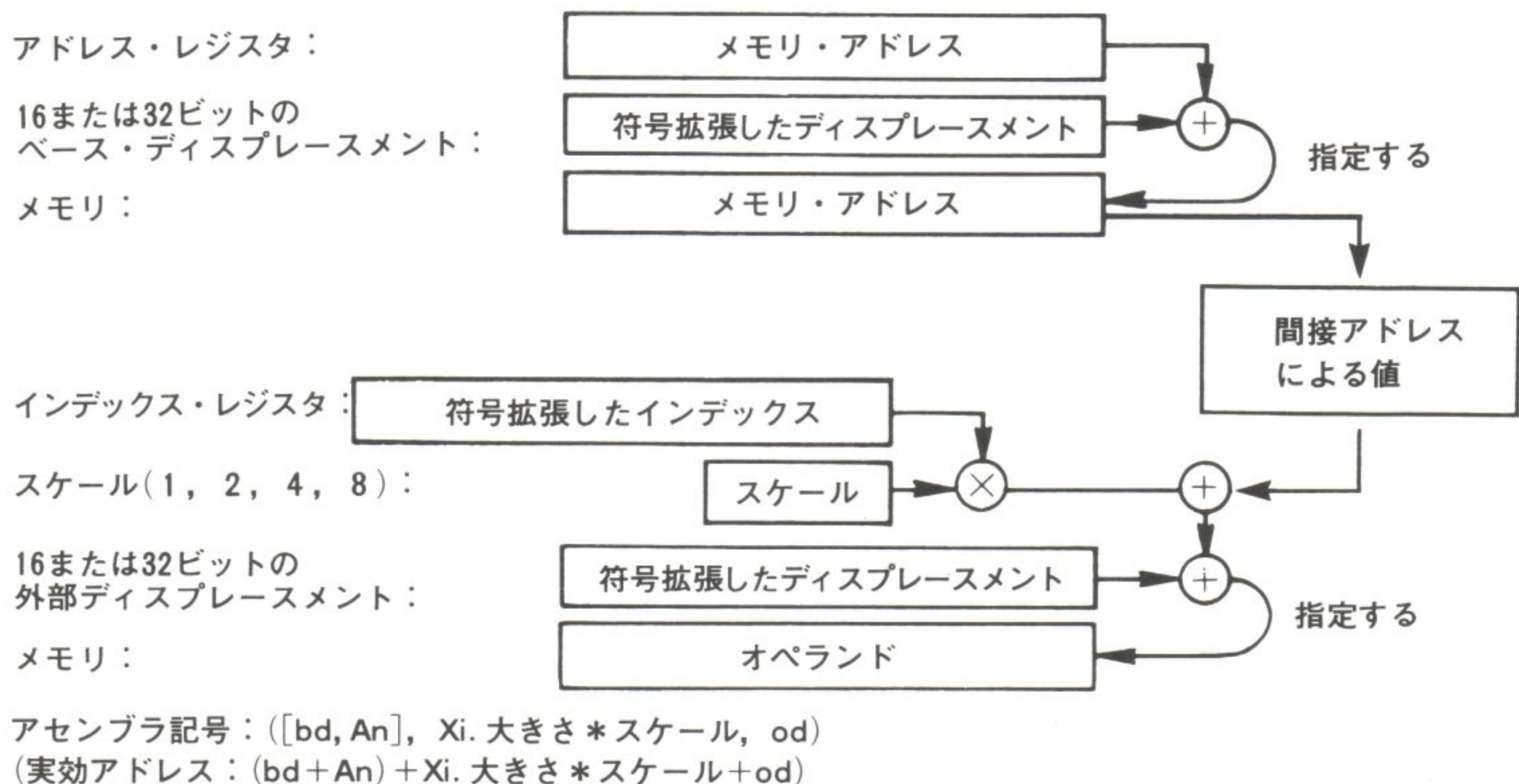
* データ・レジスタでもよいのでXiと書く．以下同様．
なお，これによってデータ・レジスタ間接もできる．

68020：　　　　(d₈，An，Xi.大きさ＊スケール)
　　　　　　　(bd，An，Xi.大きさ＊スケール)
(実効アドレス：(An)＋(Xi)＋d₈，(An)＋(Xi)＋bd)

†大きさはバイトならB，ワードならW，ロングワードならLと書く．

ポストインデックスつきメモリ間接

68020だけに使えるポストインデックスつきメモリ間接*モードでは，オペランドのアドレスは4つの値から定まる．それはアドレス・レジスタ，16ビットまたは32ビットのベース・ディスプレースメント，インデックス・レジスタの値と**スケール**の積および2番目の（外部）ディスプレースメントの内容である．

ベース・ディスプレースメントと外部ディスプレースメントの長さは16ビットか32ビットで，16ビットならば32ビットに符号つきで拡張してから用いる．インデックス・レジスタも同様で，必要に応じて加える前に符号つきで拡張する．この4つの値はオプションになっていて，省略してあれば0と仮定して実効アドレスを求めることになっている．

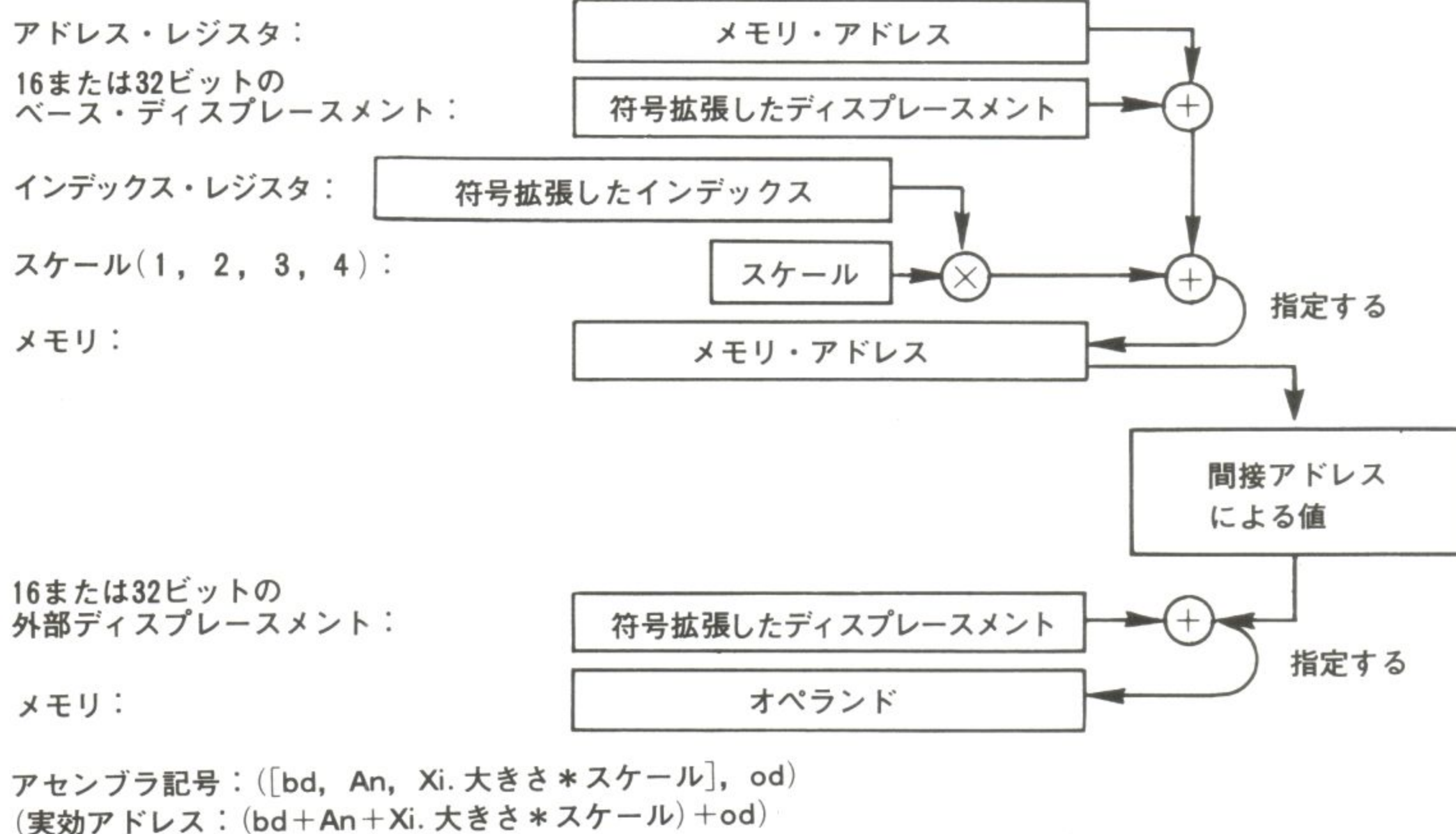
実効アドレスの計算においてはまず，アドレス・レジスタの値にベース・ディスプレースメントを加え，これを2番目の値のアドレスとして用いる．この2番目の値にインデックス・レジスタの値のスケール倍を加え，最後に外部ディスプレースメントを加えて，オペランドのアドレスが決まる．

＊　これは実は二重間接である．

プリインデックスつきメモリ間接

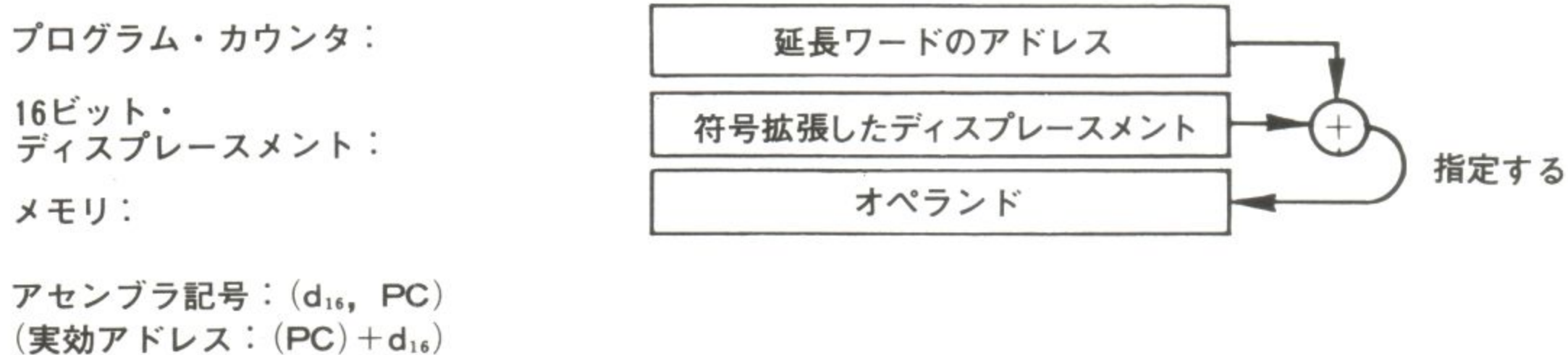
68020だけに使えるプリインデックスつきメモリ間接* モードでは，オペランドのアドレスは4つの値から定まる．それはアドレス・レジスタ，16ビットまたは32ビットのベース・ディスプレースメント，インデックス・レジスタの値と**スケール**の積および2番目の（外部）ディスプレースメントの内容である．

ベース・ディスプレースメントと外部ディスプレースメントの長さは，16ビットか32ビットで，16ビットならばそれを32ビットに符号つきで拡張してから用いる．インデックス・レジスタも同様で，必要に応じて符号つきで拡張してから用いる．この4つの値はオプションになっていて，省略してあれば0と仮定して実効アドレスを求めることになっている．

実効アドレスの計算においてはまず，アドレス・レジスタの値にベース・ディスプレースメントおよびインデックス・レジスタの値のスケール倍を加え，これを2番目の値のアドレスとする．この2番目の値に外部ディスプレースメントを加えて，オペランドのアドレスが決まる．

アセンブラ記号：（[bd，An，Xi.大きさ＊スケール]，od）
（実効アドレス：（bd＋An＋Xi.大きさ＊スケール）＋od）

ディスプレースメントつきPC間接

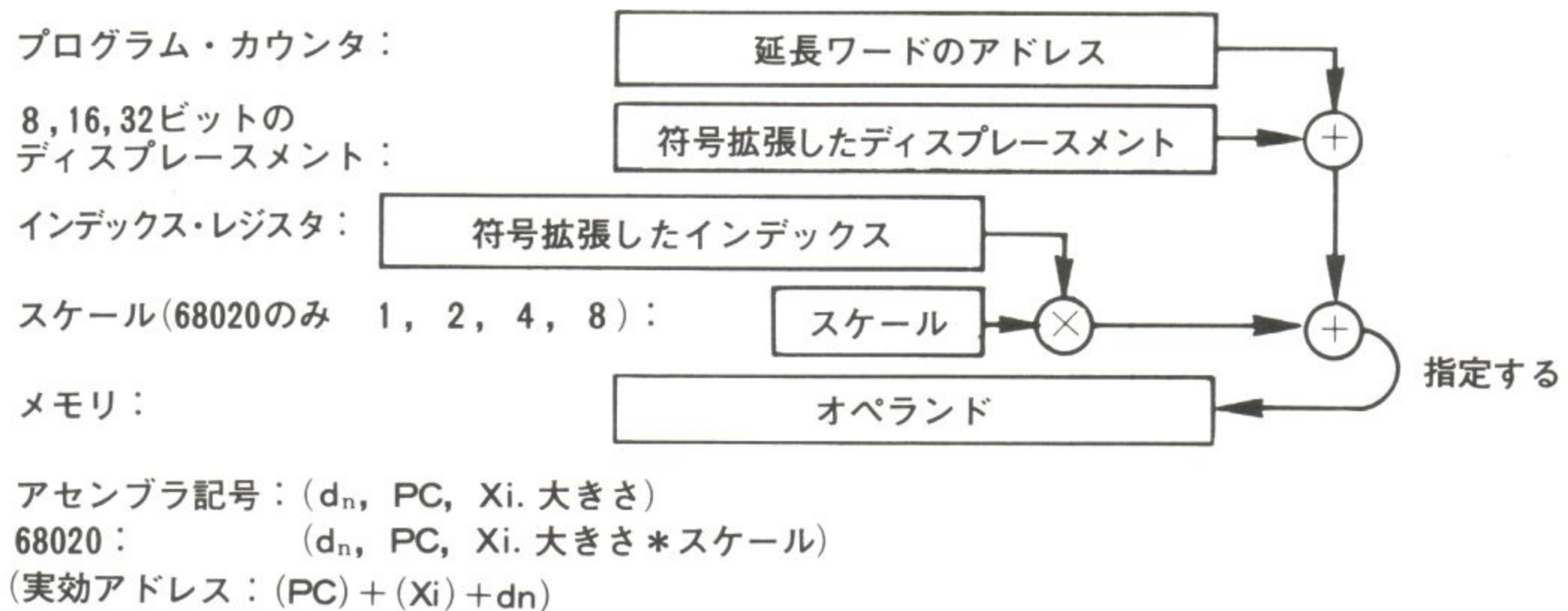
ディスプレースメントつきPC間接モードでは，プログラム・カウンタPCの値と16ビット延長ワードを符号つきで拡張した値とを加えて実効アドレスを生成する．

アセンブラ記号：（d_{16}，PC）
（実効アドレス：（PC）＋d_{16}）

＊　これも二重間接である．

インデックスおよびディスプレースメントつきPC間接

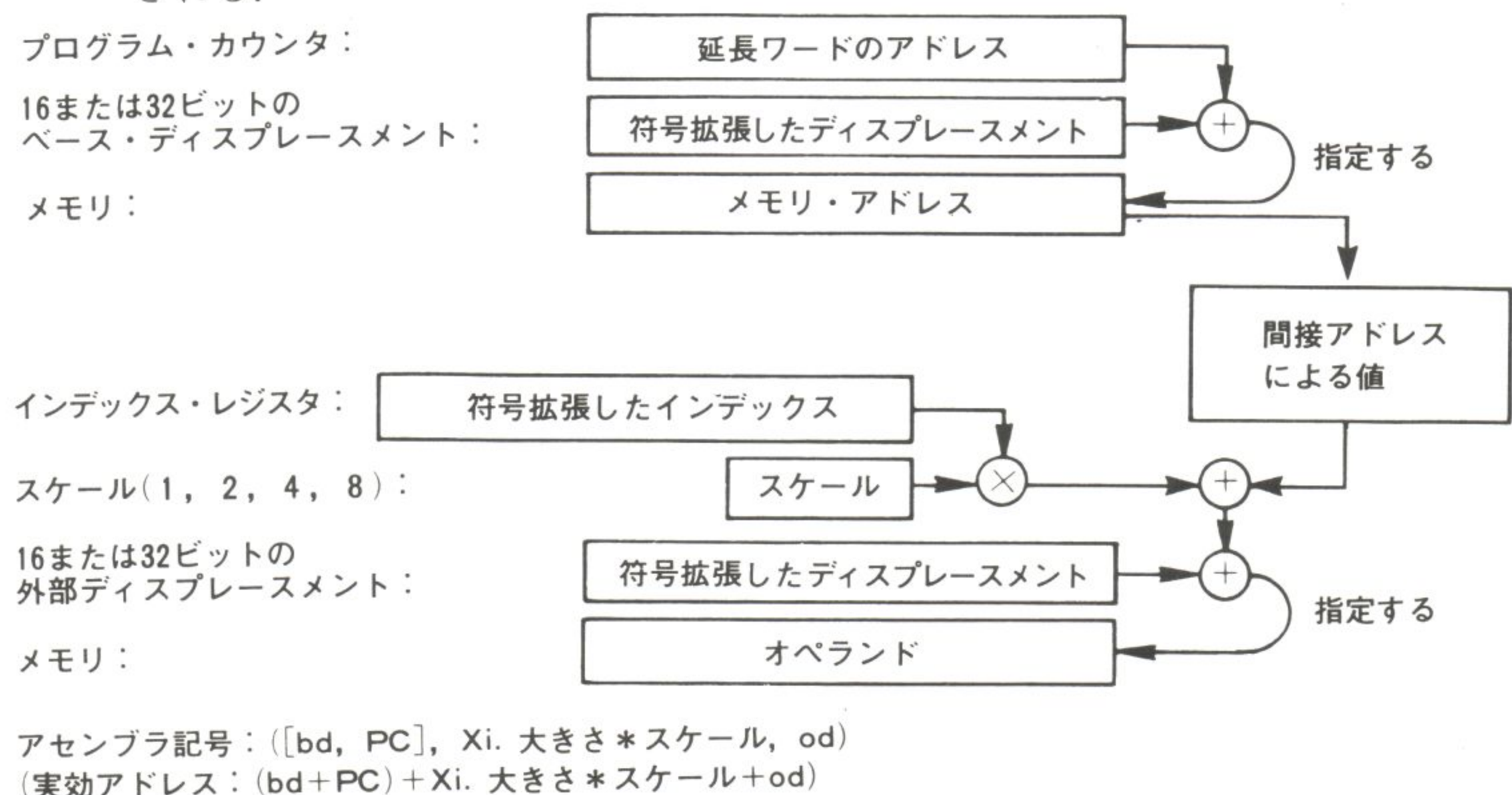
インデックスおよびディスプレースメントつき PC 間接モードでは，プログラム・カウンタの値，延長ワードの値，インデックス・レジスタの値（68020ではこれをスケール倍したもの）の和が実効アドレスになる．

この計算で，プログラム・カウンタの値は延長ワードのアドレスであり，ディスプレースメントが8ビット（68020のみ）や16ビットのときは，それを符号つきで拡張している．インデックス・レジスタの値も必要に応じて符号つきで拡張され，68020ではスケール倍される．

アセンブラ記号：（d_n，PC，Xi．大きさ）
68020：　　　　（d_n，PC，Xi．大きさ＊スケール）
（実効アドレス：（PC）＋（Xi）＋dn）

ポストインデックスつきPCメモリ間接

ポストインデックスつき PC メモリ間接*モードでは，4つの値から実効アドレスを求める．まず PC とベース・ディスプレースメントの和をアドレスとし，それにインデックス・レジスタの値をスケール倍したものと外部ディスプレースメントの値を加えてオペランドのアドレスとする．

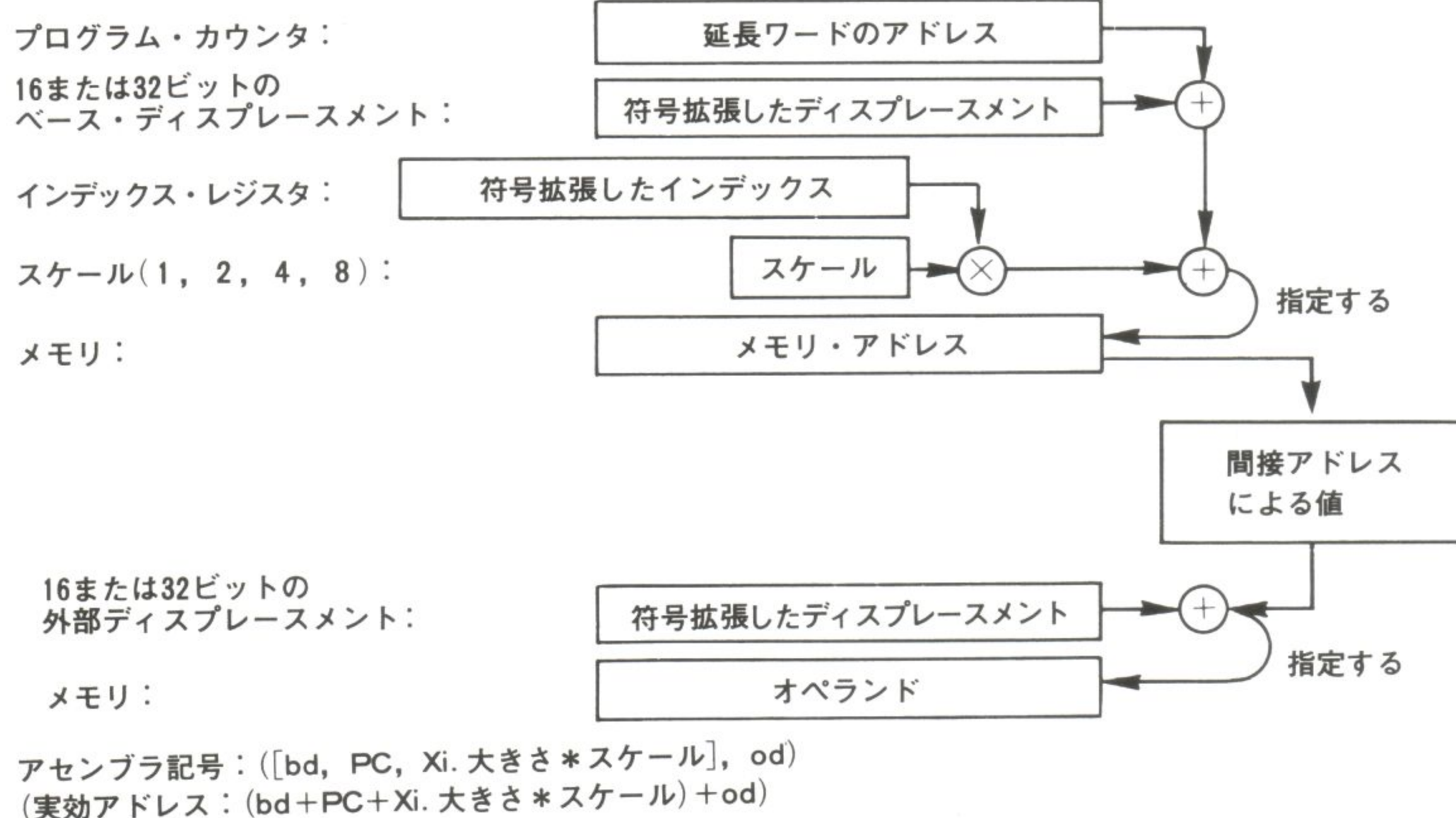
プログラム・カウンタ PC の値は，延長ワードのアドレスである．ベース・ディスプレースメントや外部ディスプレースメントのビット数は16か32であり，16ビットならば符号つきで拡張してから用いる．インデックス・レジスタの値はバイト，ワード，ロングワードのどれでもよく，必要なら符号つきで拡張する．スケールの値は1，2，4または8である．オペランドのアドレスを計算するのに必要な成分はどれもオプションでよく，省略したときは0とみなされる．

アセンブラ記号：（［bd，PC］，Xi．大きさ＊スケール，od）
（実効アドレス：（bd＋PC）＋Xi．大きさ＊スケール＋od）

＊　これも二重間接である．

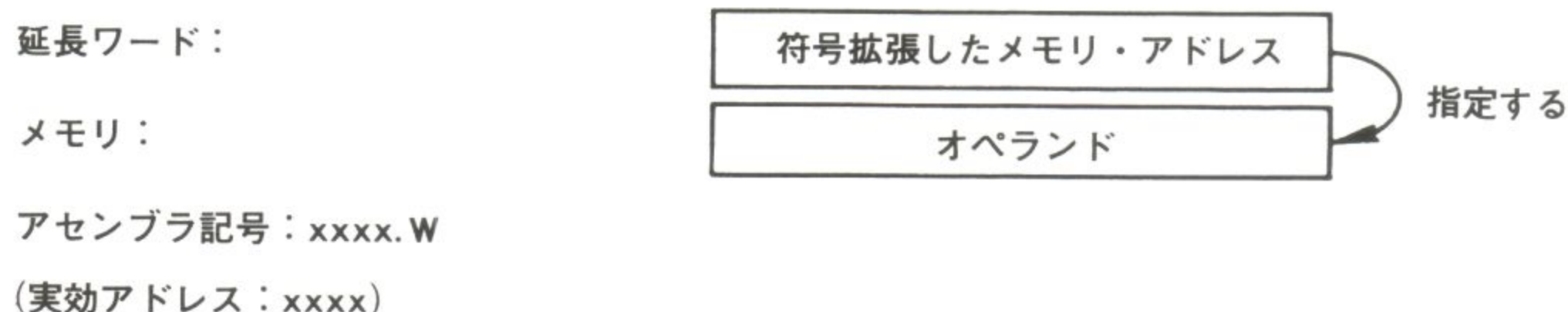
プリインデックスつきPCメモリ間接

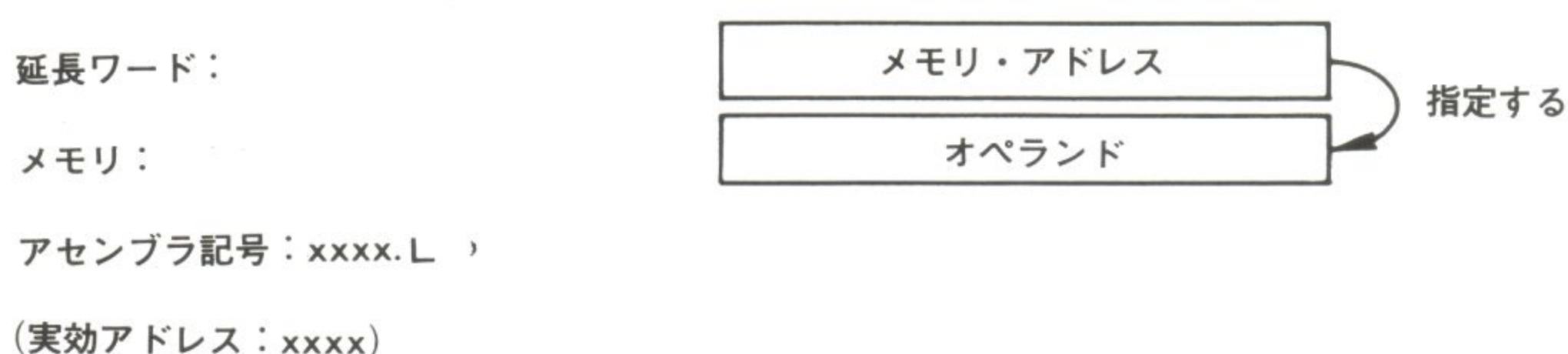
プリインデックスつきPCメモリ間接*モードでは，4つの値から実効アドレスを求める．まずPCとベース・ディスプレースメントとスケール倍したインデックス・レジスタを加えてアドレスとし，次にこのアドレスの表す値に外部ディスプレースメントを加えて，オペランドのアドレスとする．

プログラム・カウンタの値は，延長ワードのアドレスである．ベース・ディスプレースメントや外部ディスプレースメントのビット数は16か32であり，16ビットならば符号つきで拡張してから用いる．インデックス・レジスタの値はバイト，ワード，ロングワードのどれでもよく，必要なら符号つきで拡張する．スケールの値は1，2，4または8である．オペランドのアドレスを計算するのに必要な成分はどれもオプションでよく，省略したときは0とみなされる．

アセンブラ記号：([bd，PC，Xi.大きさ＊スケール]，od)
(実効アドレス：(bd+PC+Xi.大きさ＊スケール)+od)

絶対ショート

絶対ショートモードでは，命令語に続く16ビットの延長ワードを符号つきで拡張した値がオペランドのアドレスとなる．

アセンブラ記号：xxxx.W
(実効アドレス：xxxx)

絶対ロング

絶対ロングモードでは，命令語に続くロングワードがオペランドのアドレスとなる．

アセンブラ記号：xxxx.L
(実効アドレス：xxxx)

＊　これも二重間接である．

イミディエイト・データ

イミディエイト・データ・モードでは，オペランド自身が命令語に続いている．オペランドの大きさによって，命令は1ワードか2ワードの延長ワードを必要とする．

延長ワード：　オペランド

アセンブラ記号：#xxxx. 大きさ

第4章

命令セット

68K には 300 を超える固有な命令がある．しかし，大半の命令は似ており，操作対象であるデータ・タイプや命令で用いるアドレッシング・モードなどが異なるだけである．実際に 68K で用意されている基本命令は 55 個から 60 個であり，その個数はそれぞれのプロセッサによって異なる．命令セットはその変化に一貫性があり，予想できるようになっているので，比較的勉強しやすい．

命令の基本的なフォーマットは，実際上でもアセンブラのニーモニックでも同じ形をしている．すなわち実際上は，どの命令の **OP コード**も 1 ワードであり*，その中にアドレッシング・モードの一部が含まれている．アドレッシング・モードでは，ディスプレースメント，インデックス，間接などのオペランドを指定するために**延長ワード**が必要になり，命令ワード(OP コード＋延長ワード)の長さは，そのモードによって 1 ワードから 11 ワードまでとさまざまである．

上向きの互換性

Motorola 社は，68K ファミリのオブジェクト・コードのレベルで上向きの互換性を保つように配慮した．これは 68000 のコードが 68020 でも実行できることを意味する．ただしその逆は難しく，68020 のコードが 68000 で実行できる

＊ 原則として OP コードに 10 ビット用いている．

とは限らない．この考え方には意味がある．というのは，上向きに移植すればパワーと速さの点で有効であり，下向きに移植すればコストの面で役に立つからである．

命令の要約

命令には10個の基本形式がある．

- データ転送
- 整数演算
- 論理演算
- シフトおよびローテイト
- ビット操作
- ビット・フィールド操作
- 2進化10進数(BCD)操作
- プログラム制御
- システム制御
- マルチタスク/マルチプロセッサとの通信

入出力（I/O*）に関する命令はないことを注意しておく．68Kファミリでは ポートによるI/Oではなく「**メモリ割当てによるI/O**」を用いており，プロセッサはI/Oデバイスとある記憶場所で情報を交換する．プログラマやハードウェア・インターフェースのエンジニアにとっては，命令セットが単純になったといえよう．

データ転送

データ転送命令は，レジスタ同士，レジスタとメモリ，メモリ同士の間でデータを交換するもので，データの転送，アドレスの転送，レジスタのデータ交換，複数レジスタのロードとストア，スタック・フレームのリンクとアンリンクを行う．このデータ転送命令については，2つのことを注意しておきたい．一つは普通のPUSH/POP命令はなく，(A7)+，(A7)－というオート・インクリメント，オート・デクリメントを伴うMOVE命令でスタック操作を行うことである．もう一つは，他のマイクロプロセッサとは異なり，ブロック転送命令を備えていないということである．しかしDBcc(コンディション・コードによるデクリメントと分岐）命令とオート・インクリメントまたはオート・デクリメント・アドレッシング・モードを組み合わせれば，高速ブロック転送を行うことができる．

＊　Input/Outputの略．

表4.1 データ転送命令

ニーモニック	動　　作
EXG	レジスタ内容の交換
LEA	実効アドレスのロード
LINK	リンクとスタックの確保
MOVE	データの転送
MOVEA	アドレス・データの転送
MOVEC	制御コード・レジスタの転送
MOVEM	複数レジスタの転送
MOVEP	周辺装置との転送
MOVEQ	ショート・データの転送
MOVES	アドレス空間の転送
PEA	実効アドレスのプッシュ
UNLK	スタックの解放

68010と68012では「ループ可能」命令を使ってブロック転送の速さを上げ，68020では内部命令キャッシュを使ってもっと高速にできる．こういった概念はプログラマには分かりやすい．これらはこの章の最後で論ずる．**表4.1**にデータ転送命令を要約した．

整数演算

68Kには加算，減算，乗算，除算という4つの基本的な整数演算機能があり，その他，2整数の比較，整数をゼロにすること(クリア)，符号の反転，多倍長整数演算などができるようになっている．またアドレスやデータの加算，比較，減算もできる．アドレスに関する操作については16ビットと32ビットの値に制限されているが，データに関する操作については8ビットのものも含まれている．

乗除算については，符号つきと符号のないものがあり，必要に応じて使えばよい．どの68Kにおいても，16ビット整数を掛け合わせて32ビットの積が得られ，32ビット整数を16ビット整数で割って16ビットの商と16ビットの余りを得るようになっている．68020では32ビット整数を掛け合わせて64ビットの積が得られ，64ビット整数を32ビット整数で割って32ビットの商と余りが得られる．

次ページの**表4.2**に整数演算命令を要約した．

論理演算

68Kにおける論理演算として，AND(論理積)，OR(論理和)，EOR(排他的論理和)，NOT(否定*1) がある．このほか，整数を0と比較し，コンディション・コード・レジスタを設定するテスト命令も論理演算に含める．Scc命令はコンディション・コードを調べ，その結果に応じてメモリ・バイトをすべて1または0にセットする．*2

＊1　1の補数をとる命令．
＊2　16進でいえばFFか00になるということ．

表4.2　整数演算命令

ニーモニック	動　　作
ADD	データ・レジスタとの加算
ADDA	アドレス・レジスタとの加算
ADDI	イミディエイト・データとの加算
ADDQ	イミディエイト・データ（1～8）との加算
ADDX	拡張ビットXを伴う加算
CLR	オペランドのクリア
CMP	データ・レジスタとのデータの比較
CMPA	アドレス・レジスタとのデータの比較
CMPI	イミディエイト・データとの比較
CMPM	メモリ間のデータの比較
CMP 2 *	ソースの上限/下限とレジスタの比較とコンディション・コードのセット
DIVS	符号つき除算
DIVU	符号なし除算
DIVSL*	符号つき（倍）ロングワード除算
DIVUL*	符号なし（倍）ロングワード除算
EXT	レジスタにおける符号拡張
EXTB	レジスタにおけるバイトの符号拡張
MULS	符号つき乗算
MULU	符号なし乗算
NEG	反転
NEGX	拡張ビットXを伴う反転
SUB	データ・レジスタの減算
SUBA	アドレス・レジスタの減算
SUBI	イミディエイト・データの減算
SUBQ	イミディエイト・データ（1～8）の減算
SUBX	拡張ビットXを伴う減算
＊68020のみ	

表4.3に論理演算命令を要約した.

シフトおよびローテイト

68Kには（左右の）算術シフトと論理シフトおよび（左右の）拡張ビットの関係したローテイトと関係していないローテイトがある. **算術シフト**と**論理シフト**はセットされるコンディション・コードで異なっている. また右へ算術シフトすると**最上位ビット**が複製され，右へ論理シフトすると最上位ビットが0になる.

コンディション・コード・レジスタの拡張ビットXが関係しないローテイトでは，各ビットが単に左右に動き，一方の端が他方の端に回る. 拡張ビットが関係しているローテイトでは，この端のビットが拡張ビットに送り込まれ，拡張ビットの内容が他の端に送り込まれる.*

＊　キャリーCと拡張ビットXが上記の意味で関係している. Cについては上に述べていないが，これは常識の範囲に含まれる.

表4.3 論理演算命令

ニーモニック	動　　作
AND	データ・レジスタとの論理積
ANDI	イミディエイト・データとの論理積
EOR	データ・レジスタとの排他的論理和
EORI	イミディエイト・データとの排他的論理和
NOT	否定（1の補数を作る）
OR	データ・レジスタとの論理和
ORI	イミディエイト・データとの論理和
Scc	コンディション・コードのテストとセット
TST	オペランドのテストとコンディション・コードのセット

表4.4 シフトおよびローテイト命令

ニーモニック	動　　作
ASL	左方算術シフト
ASR	右方算術シフト
LSL	左方論理シフト
LSR	右方論理シフト
ROL	左方ローテイト
ROR	右方ローテイト
ROXL	拡張ビットXを伴った左方ローテイト
ROXR	拡張ビットXを伴った右方ローテイト
SWAP	上位ワードと下位ワードの交換

表4.5 ビット操作命令

ニーモニック	動　　作
BCHG	ビット反転
BCLR	ビットのクリア
BSET	ビットのセット
BTST	ビットのテスト（Zにのせる）

表4.4にシフトおよびローテイトに関する命令を要約した．

ビット操作

68K におけるビット処理命令として，各ビットのテスト，クリア，セット，反転（論理における否定）がある．クリア，セット，反転の各命令は**マルチタスク・システム**で有効である．というのは，リードとライトを一つの命令にまとめ，データ構造のアクセスがプログラムで制御できるからである．

これらの命令はしかし「不可分(indivisible)」ではない．**マルチプロセッサ・システム**において，現在実行中のプロセッサは，命令の実行中に次のプロセッサにバスを明け渡すので，プログラムは他のプロセッサのロックアウトに関係しない．不可分命令については，「マルチタスク/マルチプロセッサ命令」にある命令（表4.10）を参照すればよい．

表4.5にビット操作命令を要約した．

ビット・フィールド操作

68020には，1個のビットと同様に，一連のビット列からなる「フィールド」を処理する命令がある．これは32ビット長までのフィールドに作用し，ビットの挿入や抽出，ビット列の判定，ビットのセットやクリアや反転が行える．

表4.6にビット・フィールド処理命令を要約した．

表4.6　ビット・フィールド処理命令

ニーモニック	動　　作
BFCHG*	ビット・フィールドの反転
BFCLR*	ビット・フィールドのクリア
BFEXTS*	ビット・フィールドを符号拡張してデータ・レジスタへ転送
BFEXTU*	ビット・フィールドをそのままデータ・レジスタへ転送
BFFFO*	ビット・フィールドで最初の1のビットを探す
BFINS*	ビット・フィールドへの転送
BFSET*	ビット・フィールド全体をセット
BFTST*	ビット・フィールドのテスト
＊68020のみ	

2進化10進数(BCD)操作

68Kでは，整数と同様に**2進化10進数**(BCD)の加算，減算を行うことができる．68020ではこれらにパックBCDとアンパックBCD間の変換命令が追加されている．

表4.7にBCD処理命令を要約した．

表4.7　2進化10進数（BCD）処理命令

ニーモニック	動　　作
ABCD	BCD数の加算
NBCD	BCD数の補数の作成
PACK*	BCD数のパック
SBCD	BCD数の減算
UNPK*	パックされたデータをBCD数に戻す
*68020のみ	

プログラム制御

プログラム制御命令には，プログラム中の他のポイントへの**条件つき分岐**命令と**無条件分岐**命令とがあり，**絶対分岐**（飛越し）と**PC相対分岐**がある．また68Kにはスタックによるサブルーチン呼出しと帰還があり，68020ではサブルーチンへのパラメータ渡しのために特定の**スタック・フレーム**を認めた命令がある．**表4.8**にプログラム制御命令を要約した．

システム制御

システム制御命令は，プロセッサがスーパバイザ・モードにあるときのみ有効な特権命令，ユーザ・モードのプログラムをスーパバイザ・モードのプログラムにインターフェース*する命令，ステータス・レジスタのコンディション・

＊　インターフェースは橋渡しするという意味である．すなわち，必要なデータを送りあるいは確保し，ユーザ・モードからスーパバイザ・モードに移る．

表4.8 プログラム制御命令

ニーモニック	動　　作
Bcc	条件分岐
BRA	無条件分岐
BSR	サブルーチンへの分岐
CALLM*	モジュールの呼出し
DBcc	条件テスト，デクリメントと分岐
JMP	飛越し
JSR	サブルーチンへの飛越し
NOP	無効命令（無操作）
RTD**	帰還してスタック解放
RTE＋	例外処理からの帰還
RTM*	モジュールからの帰還
RTR	コンディション・コード復元の上の帰還
RTS	サブルーチンからの帰還
＋特権命令 ＊68020のみ ＊＊68010－68020のみ	

コード・バイト部を操作する命令を含む．

表4.9にシステム制御命令を要約した．

マルチタスク/マルチプロセッサ

マルチタスク・システムや**マルチプロセッサ・システム**では，1つのタスクやプロセッサがそれぞれ別のタスクやプロセッサをロックアウトできなければならない．これが行えなければ，あるプログラムを実行したとき，さらに他の

表4.9 システム制御命令

ニーモニック	動　　作
ANDI	イミディエイト・データとステータス・レジスタ/コンディション・コード・レジスタとの論理積
BKPT	ブレークポイントによるトラップ
CHK	データ・レジスタの境界値を調べ，大きすぎたらトラップ
CHK 2＊	レジスタの上下限を調べ，その中に入らなかったらトラップ
EORI	ステータス・レジスタとイミディエイト・データの排他的論理和
ILLEGAL	不当命令によるトラップ
MOVE	ステータス・レジスタ/コンディション・コード・レジスタとの転送
MOVEC＋	制御レジスタとの転送
MOVES＋	アドレス空間との条件つき転送
RESET＋	条件つきリセット・ラインのアサート
STOP＋	プロセッサの条件つきストップ
TRAP	無条件トラップ，例外処理の開始
TRAPcc*	条件つきトラップ
TRAPV	オーバフローによるトラップ
＋特権命令 ＊68020のみ	

表4.10　マルチタスク/マルチプロセッサ命令

ニーモニック	動　　作
CAS*	オペランドとの比較とスワップ
CAS 2*	オペランドとの比較とスワップ
cpBcc*	コプロセッサの状態による分岐
cpDBcc*	コプロセッサの状態によるデクリメントと分岐
cpGEN*	コマンドのコプロセッサへの渡し
cpRESTORE*	コプロセッサの状態の復旧
cpSAVE*	コプロセッサの状態のセイブ
cpScc*	コプロセッサの状態のテストとセット
cpTRAPcc*	コプロセッサの状態によるトラップ
TAS	オペランドのテストとコンディション・コードのセット
＊68020のみ	

プログラムをアクセスできるようにするには同じ記憶場所をアクセスしないようにすることが必要となるため，大規模なロジックが必要になろう．プロセッサ 68K には不可分な「**リード－モディファイ－ライト**」サイクルがあり，命令の実行中にバスを明け渡さないようになっている．

このほか，この命令群にはコプロセッサ・インターフェース命令がある．**コプロセッサ**としては浮動小数点ユニット，メモリ管理ユニット，I/O 制御ユニットなどがあり，68020 だけがコプロセッサをハードウェアでサポートしている．**コプロセッサ・インターフェース**の詳細については**付録B**をみられたい．

表4.10にマルチタスク/マルチプロセッサに関する命令を要約した．

命令のプリフェッチとパイプライン/ループ/キャッシュ

パイプライン

68K ファミリに含まれるプロセッサには多様な「**パイプライン**」が実現されている．このパイプラインは命令のフェッチ（とその前の命令）の実行を同時に並行処理する．それは重なり合っているので，命令を順にフェッチしてから実行するプロセッサと比べて，68K は命令の実行が速い．

68000，68008，68010，68020 は 2 ワード命令をプリフェッチする．すなわち，命令をフェッチしその解読が始まったとき，プロセッサは次のワードをメモリからフェッチする．こうして命令の実行が始まると，すでに 2 ワードがメモリからフェッチされており，プロセッサがその上のワードを用いているとき，新しいワードがメモリからフェッチされる．

命令が分岐を起こすときは，（実際にはすでにプロセッサにストアしてある）この命令に続くワードは捨てられるということに注意する必要がある．しかし普通に実行されていれば，プリフェッチによる速さの増加はこういった余計な

フェッチによる損失よりもはるかに大きい．

68020も同様だが，3ワードの**プリフェッチ**を用いる点が異なる．したがって3ワードの単一命令から連続した3つの1ワード命令までの同時並行操作ができる．

繰返し可能な命令

68010と68012ではパイプラインという考えがさらに改善されている．実行に先立ってプロセッサは2ワードをプリフェッチするが，Motorola社はその一つがDBcc(コンディション・コードによるデクリメントと分岐)であれば，これらを「スマート」に処理するようにした．プロセッサがメモリから命令を繰り返してフェッチしなくとも，**ループ命令**を実行できる場合もある．

DBccはループ・カウンタ，分岐条件，分岐先を示すディスプレースメントという3つのオペランドをもつが，それが1ワード命令に続いており，この1ワード命令の後で分岐するような特殊な場合には，このループの内側の命令をメモリからフェッチする必要がない．

表4.11に繰返し可能な命令をまとめた．

表4.11　繰返し可能な命令（68010，68012）

ABCD	CMPA	ROL
ADD	EOR	ROR
ADDA	LSL	ROXL
ADDX	LSR	ROXR
AND	MOVE	SBCD
ASL	NBCD	SUB
ASR	NEG	SUBA
CLR	NEGX	SUBX
CMP	NOT	TST
	OR	

命令キャッシュ

68020には，命令のパイプラインに加えて，チップ自身に付加された256バイトのキャッシュ・メモリがある．プロセッサはメモリから命令をフェッチしてはキャッシュにコピーする．そしてキャッシュにおけるトラック・アドレスを保持し，メモリからフェッチする前にキャッシュからフェッチする．

したがってメモリ・サイクルで命令のフェッチを待つ必要がないので，プロセッサの実行速度が上がる．マルチプロセッサ環境では，システム・バスを命令フェッチと結びつけないから，すべてのプロセッサで速くなる．

プロセッサは，オペランドをキャッシュにそのまま置いているのではないことを注意する．同じくスーパバイザ・プログラムではキャッシュを禁止することができる．これは例外処理ルーチンでは有効である．それは，例外の時点において，ユーザ・プログラムはキャッシュによらずに実行できるからである．

第5章

信号の特徴

この章では，68K ファミリのプロセッサに必要な入出力信号について述べる．このファミリのプロセッサには，似ているところもあれば異なるところもあるので，分かりやすくするために，8ビットまたは16ビットのデータ・バスを用いるもの（68000，68008，68010，68012）を最初に述べ，次に32ビット・データ・バスをもつもの（68020）について述べよう．

68K チップを供給している Motorola 社およびその他の企業ではいろいろなパッケージを出しているので，ピンと信号の対応についてはここでは触れない．それについては**付録D**を見られたい．

68000,68008,68010,68012における信号

図5.1から**図5.4**（42～43ページ）が各16ビット・プロセッサにおける信号を機能別にまとめたものである．

データ・バス　D0-D15が双方向データ・バスである（68008ではD0-D7）．

アドレス・バス　A1-A23がアドレス出力バスである（68008ではA0-A19；68012ではA1-A29とA31）．このパッケージで分かるように，68Kは，Intel社の8086，Zilog

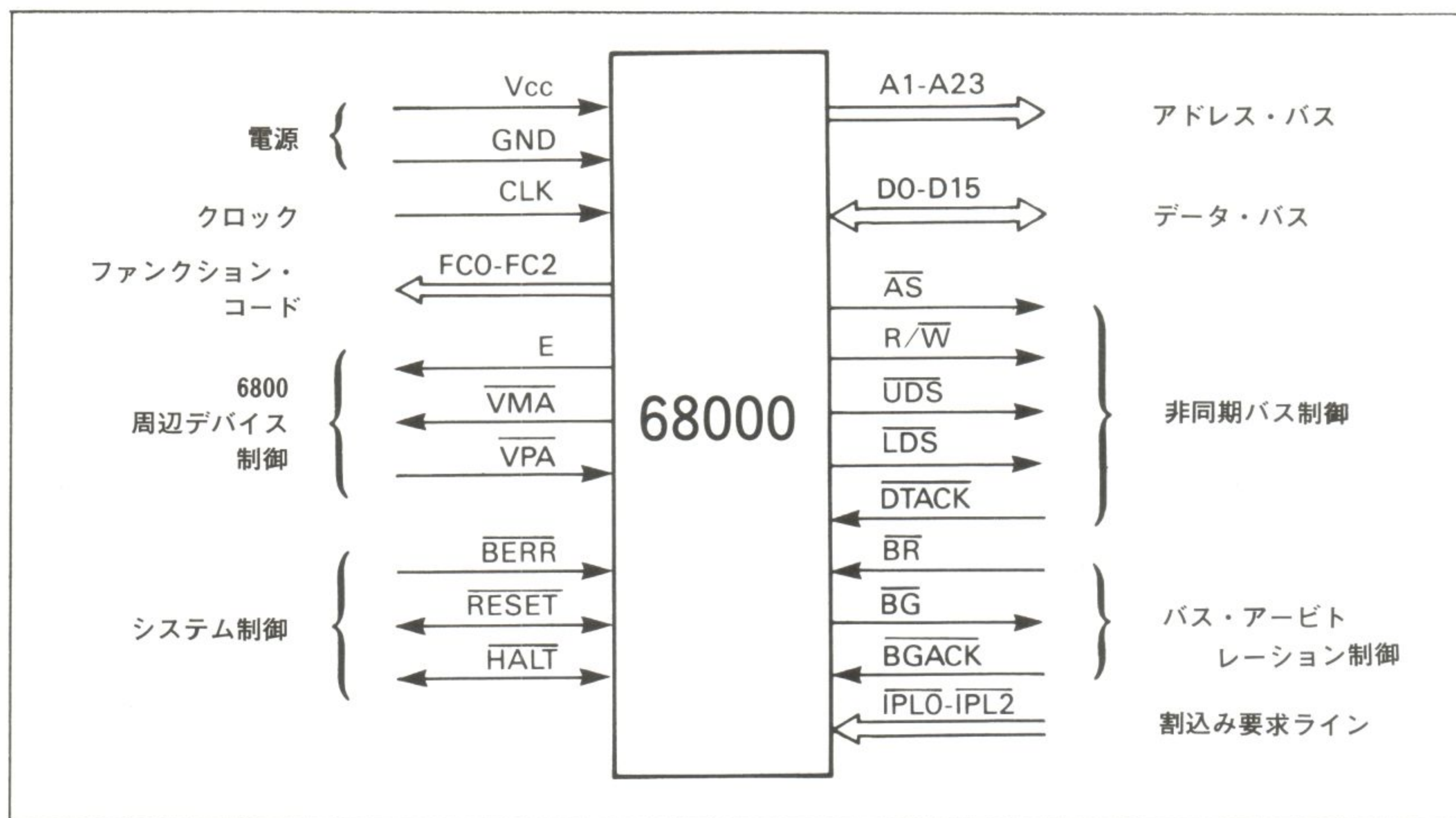



図5.1　68000における信号の機能図

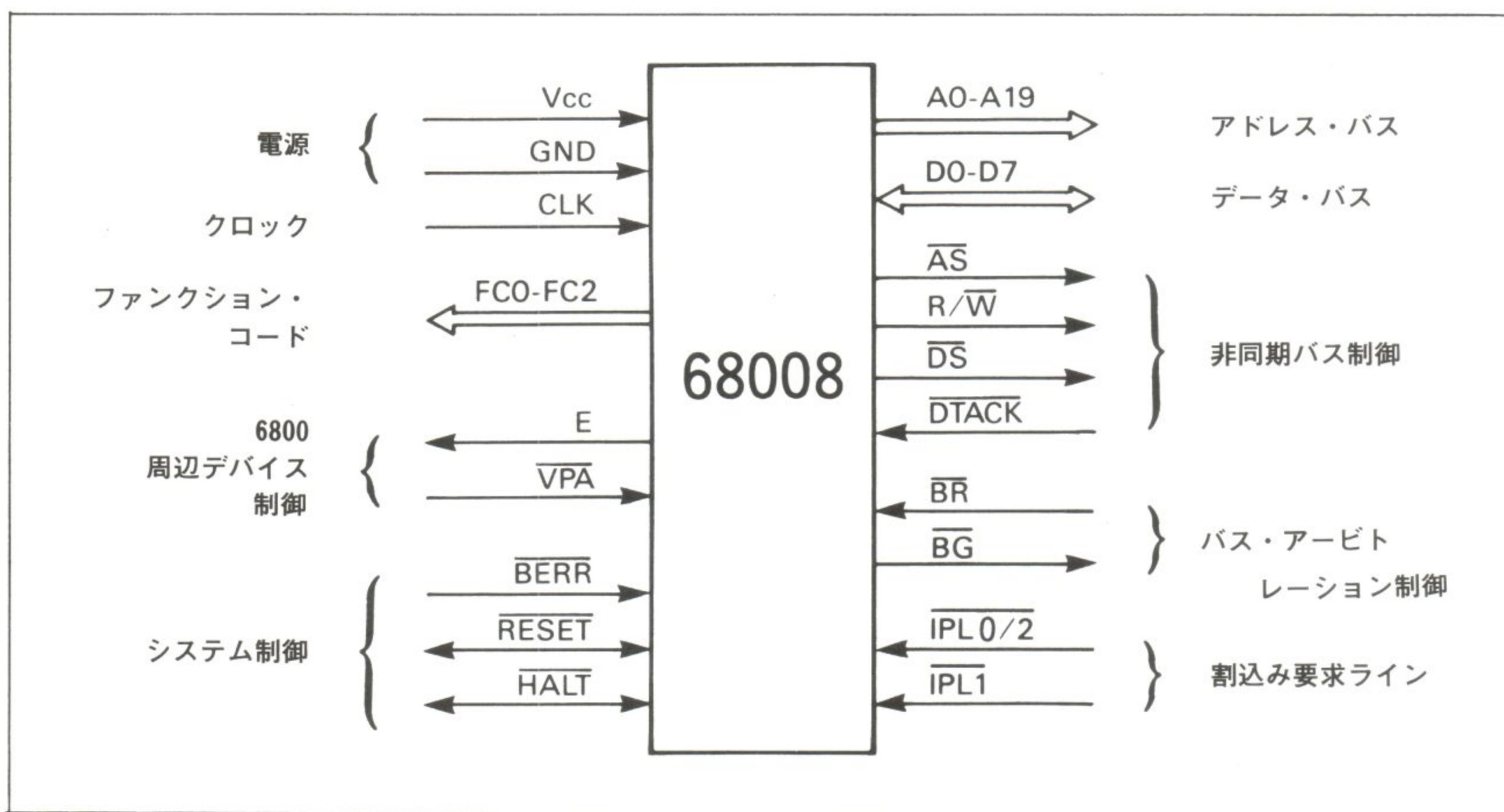



図5.2　68008における信号の機能図

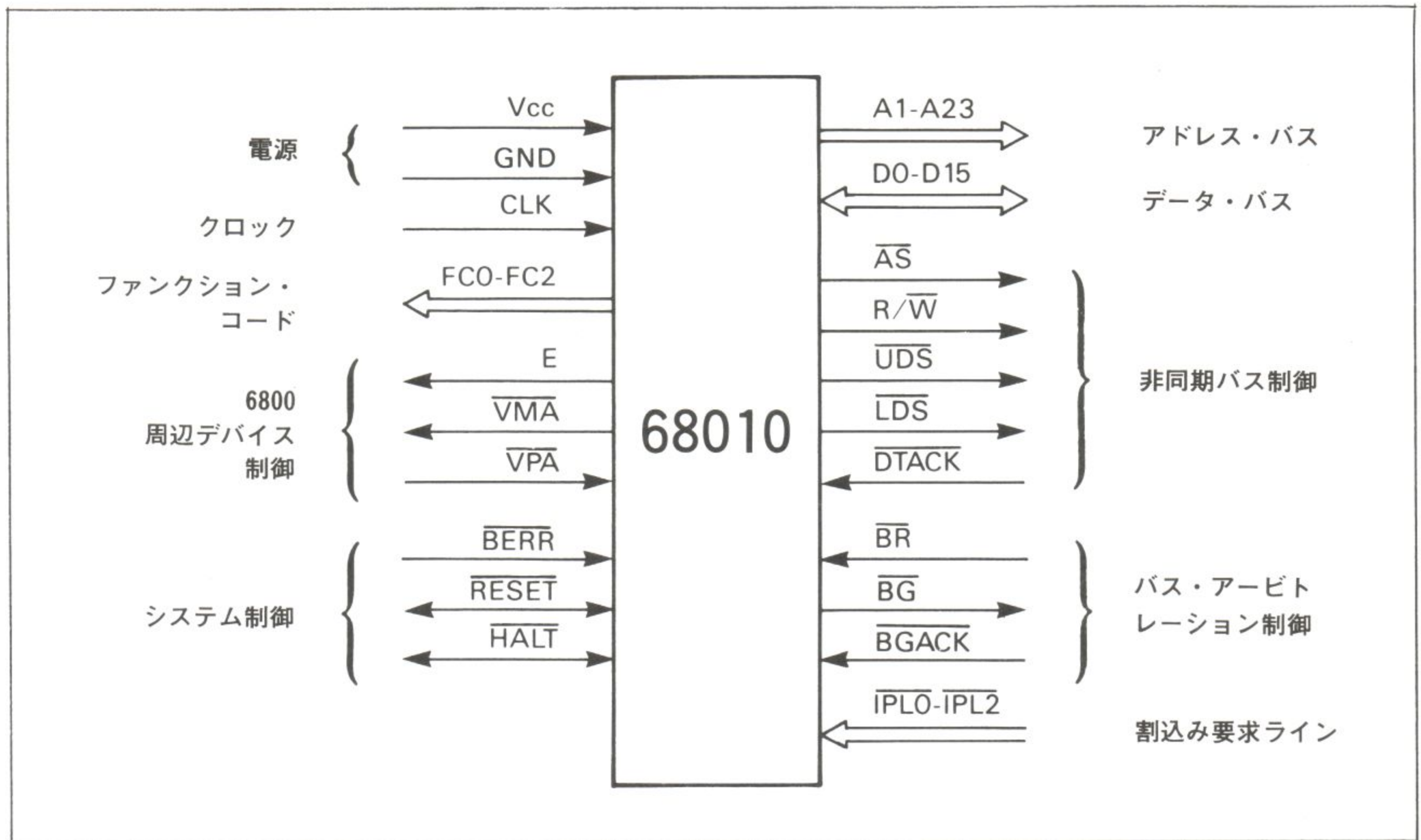



図5.3　68010における信号の機能図

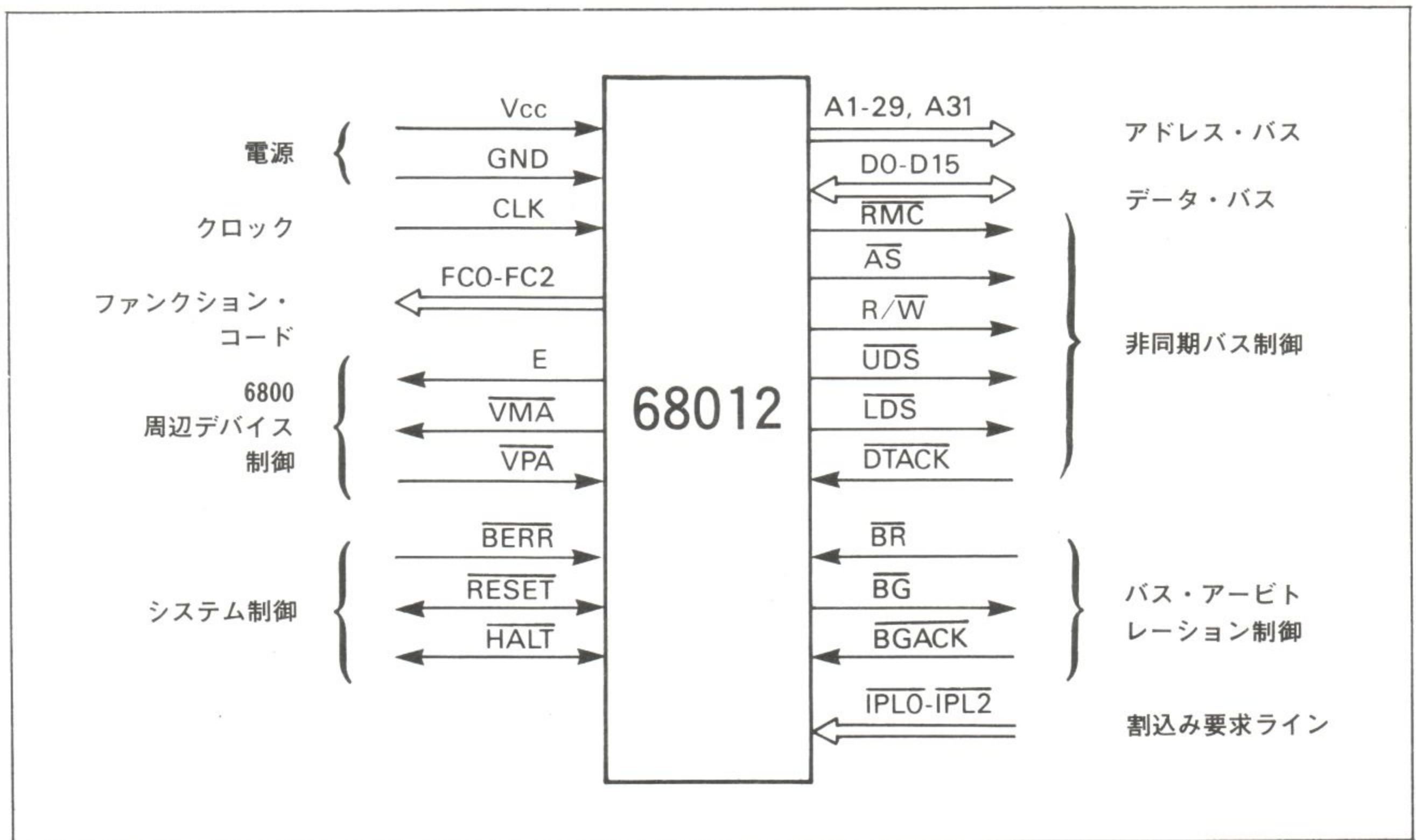



図5.4　68012における信号の機能図

社のZ8000，National Semiconductor 社のNS32000などのポピュラーな他のプロセッサと異なり，アドレス・ラインとデータ・ラインを共用していない．

実は，68008は，最下位アドレス・ライン(A0) をもつ唯一つのプロセッサである．しかもそのデータ・バス幅が8ビットなので，一度に8ビット（1バイト）のデータしかアクセスしない．

データ・ストローブ

データ・ストローブ信号 ($\overline{DS}$) は，68008のデータ・バスが使用中であることを示す．

上位/下位データ・ストローブ

他のプロセッサは**上位データ・ストローブ**($\overline{UDS}$) と**下位データ・ストローブ** ($\overline{LDS}$) を用い，データが16ビット・データ・バスの上位バイトか，下位バイトか，あるいは両方のバイトを使って転送しているかを決める．**表5.1**はデータ・バスに関して，$\overline{UDS}$ と $\overline{LDS}$ の意味を定めたものである．

表5.1　68000，68010，68012における $\overline{UDS}$ と $\overline{LDS}$

$\overline{UDS}$	$\overline{LDS}$	R/$\overline{W}$	D8-D15	D0-D7	操作
ハイ	ハイ				
ロー	ロー	ハイ	データ・ビット8-15	データ・ビット0-7	ワード・リード
ハイ	ロー	ハイ		データ・ビット0-7	バイト・リード
ロー	ハイ	ハイ	データ・ビット8-15		バイト・リード
ロー	ロー	ロー	データ・ビット8-15	データ・ビット0-7	ワード・ライト
ハイ	ロー	ロー	データ・ビット0-7	データ・ビット0-7	バイト・ライト
ロー	ハイ	ロー	データ・ビット8-15	データ・ビット8-15	バイト・ライト

（網掛け）有効なデータ入出力なし

偶数アドレスのバイト・データをアクセスするときは，プロセッサは$\overline{UDS}$ をアサートし，$\overline{LDS}$ をネゲートする．そしてデータ・バスの第8-15ビットでデータ転送が起こる．また奇数アドレスのバイトをアクセスするときは，プロセッサは$\overline{LDS}$ をアサートし，$\overline{UDS}$ をネゲートする．そしてデータ・バスの第0-7ビットでデータ転送が起こる．ワード全体をアクセスするには$\overline{LDS}$ と$\overline{UDS}$ をともにアサートすればよく，データ転送は16ビットのデータ・バス全体で起こる．

エンコードがこうなっているので，ワード・データのアクセスは偶数バイト境界から起こるだけであることが分かる．

アドレス・ストローブ信号 ($\overline{AS}$) は，ストローブ・プロセッサの出力で，アドレス・バスとデータ・ストローブがそこで有効なデータをもっていることを示している．

リード/ライト

リード/ライト信号 (R/$\overline{W}$) は，データ・バスの転送方向がリード・サイクルかライト・サイクルのいずれであるかを決める．信号がハイならリードで，ロー

ならライトである．

データ転送アクノリッジ

$\overline{\text{DTACK}}$ が**データ転送アクノリッジ**の入力信号である．メモリのような外部ロジックは，データ・バス上でデータを受け取った（ライトの場合）ことやデータ・バスに必要なデータを置いた（リードの場合）ことをプロセッサに知らせるため，$\overline{\text{DTACK}}$ をアサートしなければならない．必要なら，$\overline{\text{DTACK}}$ を受け取るまでプロセッサが「ウェイト状態」をそのサイクルに自動的に挿入する．したがって，いろいろな速さの周辺装置をアクセスすることができる．

ファンクション・コード

FC0，FC1，FC2 が**ファンクション・コード**の出力を示す．$\overline{\text{AS}}$ がアサートされているときだけ有効なこれらの出力は，**表5.2**に要約したように 68K が現在引き受けているバス動作のタイプと対応している．表からわかるように，ファンクション・コードの出力によって，メモリ・アクセスがデータ対プログラムとユーザ対スーパバイザというように分けられる．これは，68451 のような**メモリ管理ユニット**によって保護メモリと非保護メモリのアクセスが制御できることを示している．

ファンクション・コードの 5 番目のエンコードが，「CPU 空間」として定められている．前著 68000 と 68008 の解説書では，このコードを，「割込みアクノリッジ」とした．一般には割込みに対してアクノリッジを立てるため，リードやライトという普通命令の前に，プロセッサは CPU 空間を周辺装置との通信に用いる（次の割込み信号の項を参照）．

68010 と 68012 ではファンクション・コード・レジスタ（**SFC** と **DFC**）に関する MOVE 命令によって，ファンクション・コードの出力を定めることができる．

表5.2　ファンクション・コードの要点

FC2	FC1	FC0	マシン・サイクルのタイプ
0	0	0	（網かけ）
0	0	1	ユーザ・データ・メモリのアクセス
0	1	0	ユーザ・プログラム・メモリのアクセス
0	1	1	（網かけ）
1	0	0	（網かけ）
1	0	1	スーパバイザ・データ・メモリのアクセス
1	1	0	スーパバイザ・プログラム・メモリのアクセス
1	1	1	CPU 空間（割込みアクノリッジ）

（網かけ）予備，現在のところ未定義

割込み信号

周辺装置がプロセッサに，たとえば，新しいワード・データを送るため，割込みをかけたとき，周辺装置は割込みレベル・ライン $\overline{\mathrm{IPL0}}$，$\overline{\mathrm{IPL1}}$，$\overline{\mathrm{IPL2}}$ をいくつかアサートする．この3つの入力は要求した**割込みレベル**を2進数で表現している．*[1]

ステータス・レジスタには3ビットの**割込みマスク***[2]があることを前に述べたが，それは例外処理の開始に必要な割込みレベルを定めている．割込みを始めるにあたり，周辺装置はその優先度レベルを割込みライン上にアサートする．割込みマスクがこのレベルを受け取るように設定されたら，ファンクション・コード・ラインによって割込みに応答する．例外処理についてはさらに第7章で論じよう．

68008では $\overline{\mathrm{IPL0}}$ と $\overline{\mathrm{IPL2}}$ を1つの信号にまとめているので，プロセッサの割込みレベルは0，2，5，7に制限される．*[3]

バス・エラー

$\overline{\mathrm{BERR}}$ はバス・エラー入力信号である．この信号がアサートされると，68Kは例外処理シーケンスを始める．$\overline{\mathrm{BERR}}$ の目的は，外部デバイスがリード/ライト動作にうまく応答できない（すなわち $\overline{\mathrm{DTACK}}$ をアサートしない）ことをプロセッサに知らせることである．また，保護メモリへの書込みなどの不正なアクセスを**メモリ管理ユニット**（MMU）に要求したときにも，$\overline{\mathrm{BERR}}$ が用いられる．*[4]

前者の場合には，システムは $\overline{\mathrm{BERR}}$ 信号を生成する外部ハードウェアを現在もたねばならず，後者の場合には，$\overline{\mathrm{BERR}}$ 入力ラインを普通はMMUのページ・フォールト・ラインまたはそれと同等なものと結合する．

68010や68012は仮想メモリ・プロセッサなので，そこでの $\overline{\mathrm{BERR}}$ は少しようすが違う．これに関する情報は第7章で詳しく述べよう．

プロセッサ・ホールト

$\overline{\mathrm{HALT}}$ にはいろいろな機能がある．入出力信号であり，単独でも働き，また他の信号とともに働くこともある．入力信号として単独にアサートされたときは，プロセッサをインアクティブ状態にし，それは $\overline{\mathrm{HALT}}$ 信号がネゲートされるまで続く．

また $\overline{\mathrm{HALT}}$ は，たとえばダブル・バス・フォールトなどによって，プロセ

＊1　$\overline{\mathrm{IPL0}}$，$\overline{\mathrm{IPL1}}$，$\overline{\mathrm{IPL2}}$ がそれぞれ第1位，第2位，第3位の2進数になっている．

＊2　I0，I1，I2を指す．

＊3　000＝0，010＝2，101＝5，111＝7ということである．

＊4　現在実行中のバス・サイクルで問題が起こったことを，プロセッサに知らせるための信号が $\overline{\mathrm{BERR}}$ である．アサートの原因としては，デバイスの応答がないか，割込みベクタ番号を取り込めないか，メモリ管理ユニットに対する不正なアクセス要求があったかなどの原因があげられる．

ッサが例外処理を中止したことを示す出力信号としても機能する．このとき外部ロジックはこの状態を知ることができる．

$\overline{\text{HALT}}$ と $\overline{\text{BERR}}$ を一緒に用いることもできる．$\overline{\text{HALT}}$ が $\overline{\text{BERR}}$ とともにアサートされると，プロセッサは直前のサイクル[*1]を再実行する．

リセット

$\overline{\text{HALT}}$ と同じく $\overline{\text{RESET}}$ も双方向信号である．プロセッサが $\overline{\text{RESET}}$ 命令を実行すると，$\overline{\text{RESET}}$ がアサートされる．これはまた $\overline{\text{HALT}}$ と共用して入力信号にもなり，両者がアサートされると 68K は初期化される（リセット・ベクタによるトラップを含む）．

クロック

プロセッサはタイミング信号を必要としており，クロック入力信号 CLK がこれを給供する．

バス・アービトレーション信号

バス・アービトレーション信号として**バス・リクエスト**（$\overline{\text{BR}}$），**バス・グラント**（$\overline{\text{BG}}$），**バス・グラント・アクノリッジ**（$\overline{\text{BGACK}}$）がある．[*2] これらの信号をシステムで用い，DMA コントローラなど他のデバイスはバス制御が必要な**バス・マスタ**として機能できるようになる．外部デバイスは $\overline{\text{BR}}$ 信号をアサートしてシステム・バスのアクセスを求める．このとき現在のバス・サイクルの完了後に 68K はバスを解放する．

プロセッサは $\overline{\text{BG}}$ 信号を出力し，現在のサイクルの最後においてバスが利用できるようになったことを外部デバイスに知らせる．そこで $\overline{\text{BGACK}}$ 信号をアサートして，68K がバスを解放したことをデバイスが認め，その使用がすむまでデバイスは $\overline{\text{BGACK}}$ を保持する．ただし，68008 はこの $\overline{\text{BGACK}}$ 信号をもっていない．

このほかにもバスの進行制御における**ハンドシェイク**がいろいろあるが，その詳細は第 6 章で述べることとする．

6800 ファミリの信号

Motorola 社の初期のマイクロプロセッサ 6800 シリーズは 68K とは遠い関係にあるので，同社では 68K に 3 種類のバス信号を用意し，それによってシステム設計者が 6800 の周辺チップを広く利用してシステムを設計できるようにした．この 3 種類の信号が**イネーブル信号**（E），**有効周辺アドレス信号**（$\overline{\text{VPA}}$），**有効メモリ・アドレス信号**（$\overline{\text{VMA}}$）である．

68K と異なり，6800 は**同期式バス・インターフェース**を用いている．これはシステムにおけるデータ転送が**クロック信号**と同期しているという意味である．

＊1　バス・エラーを生じたバス・サイクルの意．

＊2　$\overline{\text{BR}}$ は CPU の現在のバス・サイクルの終わった時点でバス制御を解放し，バス・マスタを移す．

$\overline{\text{BGACK}}$ はバス・マスタになったことを示すもので，$\overline{\text{BG}}$ を受信し，$\overline{\text{AS}}$ と $\overline{\text{DTACK}}$ がインアクティブで他のデバイスがバス・マスタになっていないとき使える．

このクロックとして，68K の出力するE信号を用いる．Eの周波数は 68K に入力されるクロック周波数の 1/10 で，Eの周期は CLK 周期の 10 倍である．Eは6CLK サイクルの間ローになり，4CLK サイクルの間ハイになる．

6800 ファミリのデバイスは，6800 タイプのデータ転送を 68K に要求していることを知らせるため，$\overline{\mathrm{VPA}}$ 信号を用いる．システムはデバイス・アドレスを定めるために外部ロジックを用意する必要があり，このロジックが $\overline{\mathrm{VPA}}$ 信号をアサートしなければならない．

68K が $\overline{\mathrm{VPA}}$ 信号を受けるとデータ転送のタイミングが変わり，E信号と同期する．そこでプロセッサは $\overline{\mathrm{VMA}}$ 信号をアサートし，データ転送が始まる．

68008 には $\overline{\mathrm{VMA}}$ がなく，$\overline{\mathrm{VMA}}$ 信号は外部ロジックがアサートする．

6800 ファミリの周辺装置とのインターフェースについては**付録B**でもっと詳しく述べる．

68020の信号

68020 にはさらに進んだ特徴があり，入出力信号についても追加変更が加えられている．68020 の信号を見ると，信号名や機能などでその他の 68K と類似している点もあるが，データ・バスとアドレス・バスが完全な 32 ビット幅をも

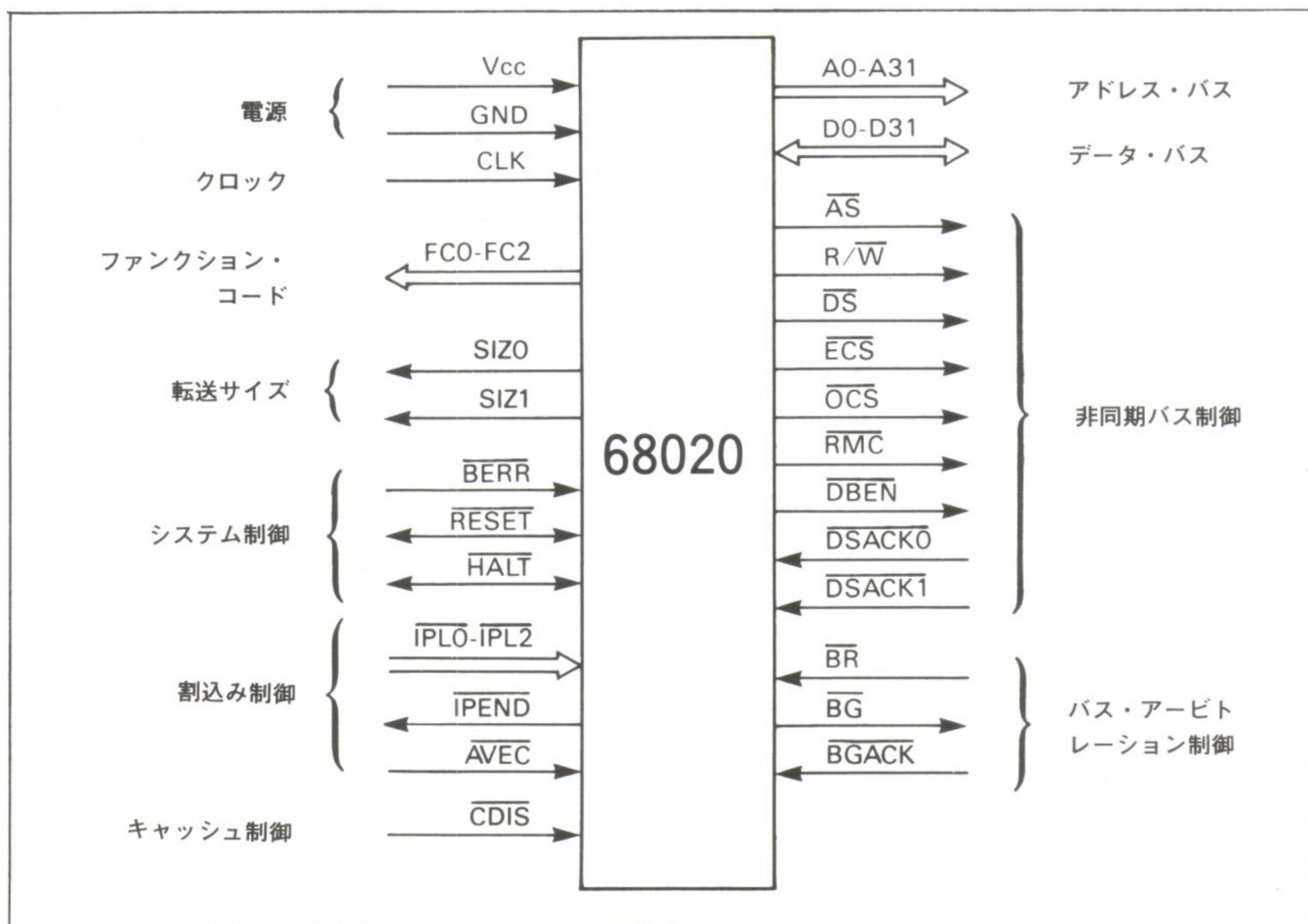



図5.5　68020における信号の機能図

ち，6800ファミリのことが考えられていない点が大きく違っている.

図5.5に68020の機能信号群を示す．**付録D**には信号とピンの関係をまとめた.

データ・バス

データ・バスは双方向で32ビットの幅があり（D0-D31），バス・サイズを動的に扱える．すなわち，データ・バスを自由に用いて，1バイト，2バイト，3バイト，4バイトのデータを送ることができる．それについては第6章で述べる.

アドレス・バス

アドレス・バスは，完全な32ビット幅をもっている（A0-A31）．A0があることに注意されたい．つまり68000や68010などにはA0がなかったことを確認して欲しい．この大きなアドレス・バスにより，4ギガバイトの強力なアドレス空間ができる.

転送サイズ

データ転送におけるデータ・バスの幅を制御する方法として，転送サイズ信号SIZ0，SIZ1がある．これらはデータ転送の際のバイト数を定めるのに使う．第6章でサイズ信号の使い方とデータ転送の詳細を述べる.

リード/ライト

68020も他の68Kと同じく，周辺装置との非同期式インターフェースを操作するために，多数のバス制御信号を用いている．**リード/ライト信号**（$R/\overline{W}$）はその1つで，データ転送の向きを示し，このラインには真の状態が2つある．真のハイはリードを意味し，真のローはライトを示す.

データ・ストローブ

データ・ストローブ（$\overline{DS}$）信号は，データ・バスが使われているかどうかを示す．リード・サイクルでアサートされていれば，周辺装置がデータ・バスをロードしていることを示す．ライト・サイクルでアサートされていれば，周辺装置がバス上にデータをラッチしていることを示す.

アドレス・ストローブ

アドレス・ストローブ（$\overline{AS}$）は出力信号で，プロセッサが有効なファンクション・コード，アドレス情報，サイズ，リード/ライトの状態に関する情報をそれぞれのライン上に配置したことを示す.

ファンクション・コード

ファンクション・コード信号（FC0，FC1，FC2）により現在のプロセッサの状態と必要なアドレス空間とを同一視できる．FC0-FC2を表5.2に示したように組み合わせると，データ領域かプログラム領域か，ユーザ・アドレス空間かスーパバイザ・アドレス空間かがわかる.

その5番目の組み合わせが「**CPU空間**」である．4つのサブタイプがこのCPU空間に含まれており，その中にブレークポイント・アクノリッジ，モジュールに関する命令，コプロセッサとの通信，割込みアクノリッジがある．プロセッサは，アドレス・バスを用いてこれらのモードを区別するとともに，周辺装置へもっと詳しい情報を送る．次ページの**図5.6**にCPU空間のアドレス・バスのエンコードを示した.

スーパバイザ・ルーチンでは，MOVES命令を用いて，**ソースおよびデスティネーション・ファンクション・コード・レジスタ（SFC，DFC）**にファン

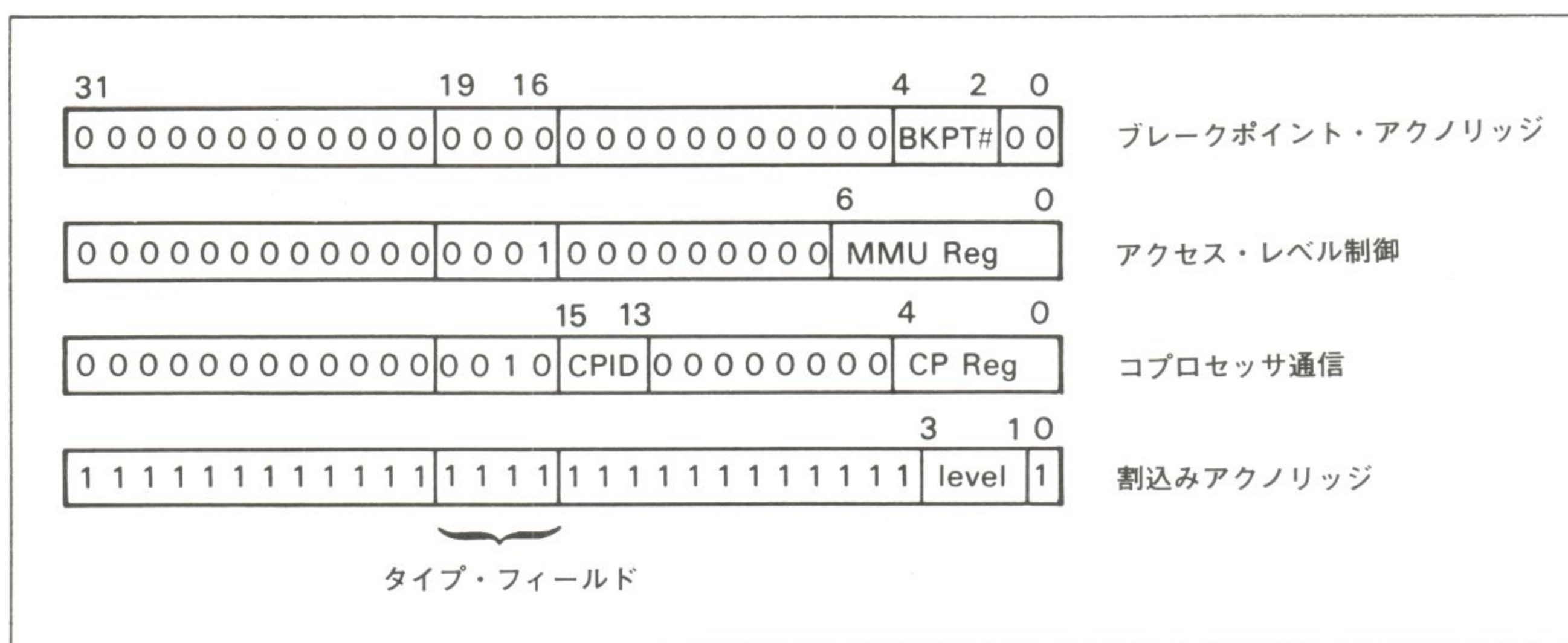



図5.6　CPU空間のアドレス・バスのエンコード

クション・コードを設定することにより，特定のアドレス空間を要求することができる．

データ転送とサイズ・アクノリッジ

これらの入力信号（$\overline{\text{DSACK0}}$，$\overline{\text{DSACK1}}$）は，データ転送の完了をプロセッサに示すものである．リード・サイクルでは，これは周辺装置がデータ・バス上にデータを置き，プロセッサがそれをラッチしたことを意味する．ライト・サイクルでは，これは周辺装置がデータを読み込み，プロセッサがデータを出し続けている*1ことを意味する．第6章ではプロセッサと周辺装置のハンドシェイクに関する情報とACKラインが1本ではなく2本ある理由について述べる．

サイクル・スタート

プロセッサと周辺装置の間の通信をもっとうまく制御するため，68020は**サイクル・スタート**出力信号を2つもっている．プロセッサは外部サイクル・スタート信号（$\overline{\text{ECS}}$）を各バス・サイクルの最初の1/2クロックだけアサートする．オペランド・サイクル・スタート信号（$\overline{\text{OCS}}$）の機能も，オペランド転送の最初のバス・サイクルでのみプロセッサがアサートする点を除けば同様である．*2

リード-モディファイ-ライト信号

この出力信号（$\overline{\text{RMC}}$）によりバスが外部制御からロックアウトされたことが分かる．68020の命令でこの信号が有効なものはTAS（テスト・アンド・セット）命令とCAS，CAS2（コンペア・アンド・スワップ）命令に限る．このバスロック能力によりマルチプロセッサ・システムにおけるデータ完全性が確実になった．

データ・バッファ・イネーブル

データ・バッファ・イネーブル出力信号（$\overline{\text{DBEN}}$）がデータ転送における外部データ・バッファを可能にした．これにより外部バッファ・コンテンション*3がどうであろうとR/$\overline{\text{W}}$信号を変更できる．

＊1　68Kはライト動作中はデータを出し続ける．

＊2　これは三相出力信号で，タイミングもECSと同じである．

＊3　コンテンションはライン制御の方法で，複数端末が同時に接続されている状態で，バッファの待ち行列を使ってバッファを管理することを指す．

キャッシュ・ディスエーブル

キャッシュ・ディスエーブル入力信号（$\overline{\mathrm{CDIS}}$）により，オンチップ命令キャッシュが使えなくなる．

割込み要求

68020には7種の割込みレベルがある．割込み要求入力ライン（$\overline{\mathrm{IPL0}}$，$\overline{\mathrm{IPL1}}$，$\overline{\mathrm{IPL2}}$）が周辺装置からプロセッサへの割込み要求を示し，この要求を受けた後で，プロセッサはステータス・レジスタにおける割込み優先度マスクとこのラインを比較する．割込み処理についてはさらに第7章で述べる．

割込み保留

割込み保留出力信号（$\overline{\mathrm{IPEND}}$）は，プロセッサが，$\overline{\mathrm{IPL0}}$-$\overline{\mathrm{IPL2}}$をコード化し，ステータス・レジスタの割込みマスクにストアしたレベルより高い割込みレベルを受け取ったか，またはマスク不能割込みを受け取ったことを示している．

オートベクタ

プロセッサが割込み信号に応答すると，割込みを要求したデバイスは**ベクタ・テーブル**の入口を示したベクタ番号を復元する．そしてプロセッサはそれぞれの割込みルーチンを見つけることができる．また68020ではオートベクタ入力ライン（$\overline{\mathrm{AVEC}}$）による「オートベクタリング」ができる．

$\overline{\mathrm{AVEC}}$がアサートされると，ベクタを特定してデバイスを待つよりも，ベクタ・テーブルのオートベクタ*を利用してプロセッサを誘導する．以前の68Kプロセッサはオートベクタリング機能を要求するため$\overline{\mathrm{VPA}}$を用いた．第7章でさらにオートベクタリングについて述べる．

バス・エラー

$\overline{\mathrm{BERR}}$がバス・エラー入力信号である．この信号がアサートされると，68020は例外処理シーケンスに入る．$\overline{\mathrm{BERR}}$の目的は，外部デバイスがリード/ライト動作に応答しなかったこと（すなわち，$\overline{\mathrm{DSACK0}}$と$\overline{\mathrm{DSACK1}}$がアサートされなかったこと）を，プロセッサに知らせることである．このほか，$\overline{\mathrm{BERR}}$はプロセッサが保護メモリをアクセスしようとしたことをメモリ管理ユニット（MMU）に知らせるためにも使われる．

前の場合には$\overline{\mathrm{BERR}}$を生成するため，システムはそのための外部ハードウェアを提示しなければならない．後者の場合には，$\overline{\mathrm{BERR}}$入力ラインは，MMUのページ・フォールト・ライン，またはそれと同等なものに結合されるのが普通である．

プロセッサ・ホールト

$\overline{\mathrm{HALT}}$信号はいろいろな機能を実行する．それは入力信号でもあり，出力信号でもあり，またそれだけでも動作し，他のラインと共同でも動作する．

入力ラインとして単独にアサートされた場合は，$\overline{\mathrm{HALT}}$信号がネゲートされるまでプロセッサをインアクティブ状態にする．

$\overline{\mathrm{HALT}}$はまたプロセッサに命令実行を中止させる出力信号としても機能する．たとえば，ダブル・バス・フォールトがそうである．外部ロジックはそのとき，

---

* ベクタ番号では25から31で，レベルは1から7にわたる．

この状態を知ることができる．

$\overline{\mathrm{HALT}}$ と $\overline{\mathrm{BERR}}$ を一諸に使うこともできる．同時に $\overline{\mathrm{HALT}}$ と $\overline{\mathrm{BERR}}$ をアサートすると，プロセッサは直前のバス・サイクルをもう一度実行する．

リセット

$\overline{\mathrm{RESET}}$ は $\overline{\mathrm{HALT}}$ と同じく双方向信号である．プロセッサが $\overline{\mathrm{RESET}}$ 命令を実行すると，このラインをアサートする．$\overline{\mathrm{RESET}}$ はまた入力ラインとしても機能し，アサートされているときは，リセット・ベクタを経由して 68020 でトラップを引き起こす．

クロック

プロセッサが機能するためには，タイミング信号が必要である．これをクロック入力信号 CLK で示す．

バス・アービトレーション

$\overline{\mathrm{BR}}$（**バス・リクエスト**），$\overline{\mathrm{BG}}$（**バス・グラント**），$\overline{\mathrm{BGACK}}$（**バス・グラント・アクノリッジ**）がバス・アービトレーション信号である．これらの信号は，DMA コントローラのような他のデバイスが，バスの制御を必要とするバス・マスタとして機能するシステムで使用される．外部デバイスは $\overline{\mathrm{BR}}$ 信号をアサートすることによって，システム・バスへのアクセスを要求する．この後，実行中のバス・サイクルの完了を待って，68020 はバスを解放する．

プロセッサは $\overline{\mathrm{BG}}$ 信号を出力して，バスが現在のサイクルの終わりに使用可能になることを要求デバイスに知らせる．そこで $\overline{\mathrm{BGACK}}$ 信号のアサートにより，68020 がバスを放棄することをデバイスに応答する．デバイスはこのバス・アクセスがすむまで $\overline{\mathrm{BGACK}}$ を保持している．

バスの使用において，もっと多くの**ハンドシェイキング**が行われていることを注意する．これについては第6章で詳細を述べよう．

第6章

タイミングとバスの動作

68K の基本的タイミングは非常に単純であり，命令の実行は内部サイクルとバス・アクセス・サイクルを組み合わせて構成されている．本章では，マイクロプロセッサ・システムにおけるデータの入力，出力，転送で必要なバス・アクセス・サイクルについて述べる．

どの 68K プロセッサも，その外部にあるデータを**「非同期」転送**を用いて転送する．これには，データ転送を調整するために，プロセッサと周辺*が**「ハンドシェイク」**・ラインを使用するという意味がある．非同期転送を使用して，いろいろな速さの周辺とプロセッサの間でデータ転送が行える．たとえば，メモリも周辺の一種なので，高価格の高速メモリをキャッシュに用い，それよりも遅い低価格の大容量 RAM を一般メモリに用いた 68K システムも多い．

プロセッサには，リードおよびライトという基本操作がある．周辺がデータ・バス上にデータを置くときリードといい，周辺がデータ・バスからデータを受け取るときライトという．記憶場所（メモリ・ロケーション）の間でのデータ転送は MOVE 命令で処理できるが，一般にはプロセッサのレジスタがデータ

---

* 周辺とはメモリや I/O デバイスをいう．

を送受信するもとになっている.*

前章と同じく，68K の信号について述べる場合は，プロセッサを8ビットまたは16ビットのデータ・バスをもつもの（68000，68008，68010，68012）ともたないもの（68020）とに分けることができる．これには，おもな理由が2つある．すなわち，68020では**動的バス幅設定**（dynamic bus sizing）とオペランドの**不当整列**（misalignment of operand）が認められているからである．

以下において，リード/ライトの基本タイミングを分けて述べ，それに続いてホールトとストップ，システムのリセット，バス・アービトレーションなどの共通機構をまとめる．

68000-68012におけるタイミングとバスの動作

リード・タイミング

ワードリード動作のタイミングを**図6.1**に示す．以下の説明では，各**クロック・ピリオド**を2つの状態（ステート；state）に分け，S*n*のように書くこと

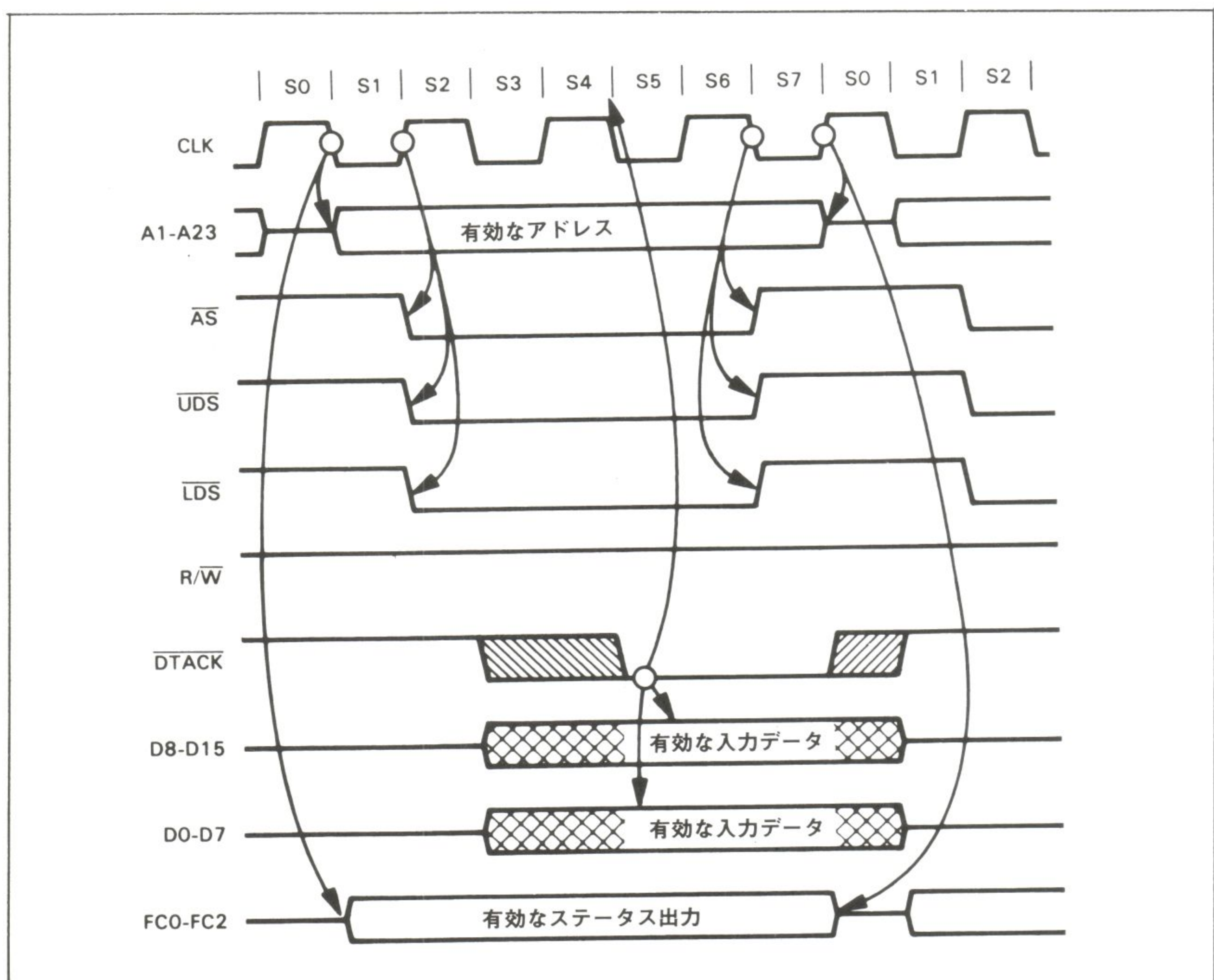



図6.1　ワード・リードのタイミング（68000，68010，68012）

＊　I/O デバイスとのデータ転送でも，一般に MOVE 命令を使う．

とする．サイクルの前半ではnは偶数で，後半では奇数である．

ワード・リード・サイクルの状態0（S0）では，アドレス・バスとデータ・バスが高インピーダンス状態にある．68Kはこの時点ではシステム・バスを用いていない．S1の始めに，周辺の位置に関するアドレス情報がアドレス・バス上に出力され，ファンクション・コード（FC0-FC2）が有効なプロセッサ・サイクルのステータス情報を示す．S2の始めに**アドレス・ストローブ**（$\overline{\mathrm{AS}}$）がアサートされ，外部ロジックではそれを使ってアドレス・バス上の現在の情報をラッチすることができる．

同時に，**上位データ・ストローブ**（$\overline{\mathrm{UDS}}$）と**下位データ・ストローブ**（$\overline{\mathrm{LDS}}$）がアサートされ，*1 16ビット・ワードの上位バイトと下位バイトをどれでも選ぶことができる．この信号は実際にはデータの「ストローブ」ではなく，厳密には16ビット・ワードの上位バイトや下位バイトの一方または両方を選ぶメモリ選択信号と考えるべきである．R/$\overline{\mathrm{W}}$は通常はアサートされており，この出力はリード・サイクル中は変わらない．

さて，68Kはデータ・バス上にデータを出力するため，アドレス指定周辺*2を待つ．周辺の準備が完了したら，68Kにデータ・アクノリッジ信号（$\overline{\mathrm{DTACK}}$）を送らなければならない．68Kは$\overline{\mathrm{DTACK}}$をS5まで待機しているが，$\overline{\mathrm{DTACK}}$が提示されないと，プロセッサは「**ウェイト状態**」（SW）をタイミング・サイクル中に自動的に挿入する．

周辺が$\overline{\mathrm{DTACK}}$をアサートすると，リード・サイクルがS5から再開される．S6の終わりに，信号$\overline{\mathrm{AS}}$，$\overline{\mathrm{UDS}}$，$\overline{\mathrm{LDS}}$がネゲートされ，プロセッサはデータをデータ・バスから内部レジスタにラッチする．外部デバイスは，このネゲートを，プロセッサがデータを受け取ってしまったので，データをバスから取り去ってもよいという意味の信号として，使うことができる．システム内の信号の歪みを許容するため，68Kは，アドレス情報とファンクション・コード情報を，S7の終わりまで維持する．

68Kがデータ・バスからデータを取り込んだことを，$\overline{\mathrm{AS}}$，$\overline{\mathrm{UDS}}$または$\overline{\mathrm{LDS}}$のネゲートで外部デバイスが感知すると，次のバス・サイクルの開始を妨げないようにするため，周辺はすぐに$\overline{\mathrm{DTACK}}$をネゲートしなければならない．

ウェイト状態

図6.1に示したとおり，リード動作でのウェイト状態は，S4とS5の間に挿入される．周辺が$\overline{\mathrm{DTACK}}$をアサートするまでのウェイト状態では，68Kは有効なアドレス出力を維持し，$\overline{\mathrm{AS}}$，$\overline{\mathrm{UDS}}$，$\overline{\mathrm{LDS}}$をアサートし続ける．68Kの動

＊1　68008では$\overline{\mathrm{DS}}$にまとめられている．ここではバイト・データがおもである．

＊2　メモリ・マップを使ったI/Oなので，周辺にはアドレスが割り当てられている．

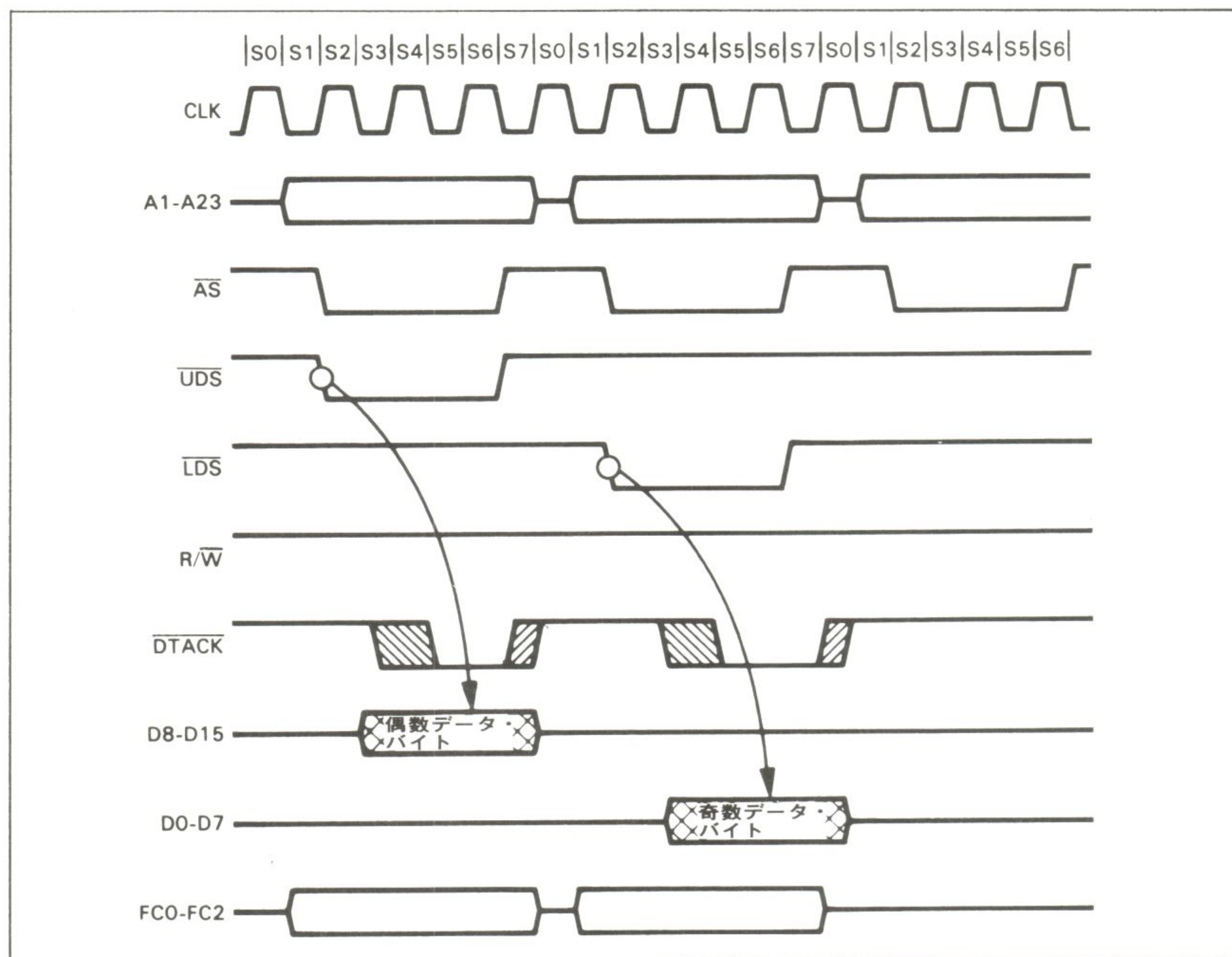



図6.2　バイト・リードのタイミング（68000，68010，68012）

作がすべてクロックに基づき，1クロック・サイクルには2つの状態があるので，ウェイト状態は偶数個挿入される．

バイト・リード・タイミング

バイト・リード動作のタイミングを，**図6.2**に示す．図に示すように，68Kは偶数データ・バイトを読んでから奇数データ・バイトを読む．これから分かるように，このタイミングと図6.1に示したワード・リードのタイミングを比べると，$\overline{\text{UDS}}$か$\overline{\text{LDS}}$の一方だけがアサートされ，データ・バスを8本だけ使っている点で，異なっているだけである．偶数アドレスにおけるバイト・リードでは，$\overline{\text{UDS}}$がアサートされデータをD8-D15に置く．奇数アドレスにおけるバイト・リードでは，$\overline{\text{LDS}}$がアサートされデータをD0-D7に置く*.

68008でのリード・タイミング

68008はデータ・ストローブ（$\overline{\text{DS}}$）を1本しかもたず，データ・バスも8本（D0-D7）だけである．これらを除けば，68008はバイト・リードに関して，他のプロセッサと同様に機能する．**図6.3**に68008のリード・サイクルを示した．

＊　図6.2で68Kがいつも2バイトを続けて読むものと速断してはならない．ここでは，両方のタイミングを説明するために，2つのリード動作を続けて示したに過ぎない．

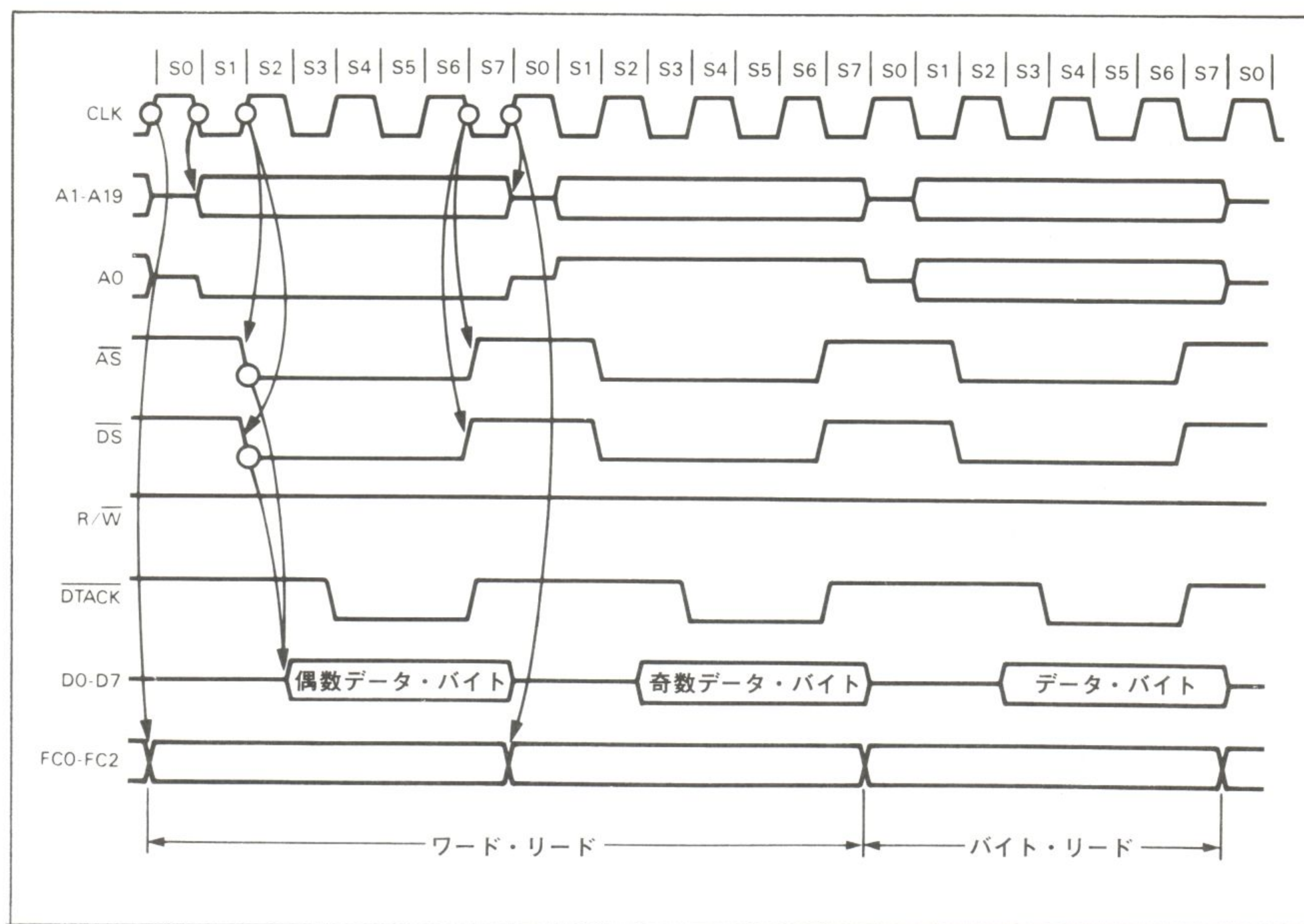



図6.3　ワードとバイトのリード・タイミング（68008）

ライト・タイミング

ワード・ライト動作のタイミングを次ページの**図6.4**に示す．リード動作の場合と同じく，メモリやI/Oデバイスなどの周辺アドレスが適当なファンクション・コードとともにS1で出力される．68Kがその前のサイクルでデータ・バスを使っていた場合は，プロセッサはすべてのデータ出力を高インピーダンス状態に戻し，そのあとライト・サイクルであることを示すために，アドレス・ストローブ（$\overline{AS}$）をアサートし，リード/ライト信号（R/$\overline{W}$）にロー・パルスを出力する．

すなわち，ここでもアドレスのラッチに$\overline{AS}$を用いることができ，68Kのデータ・バス上にデータが置かれる状態を，R/$\overline{W}$信号が周辺に示している．S3の始めにおいて，68Kがデータ・バス上にデータを出力するまでは，信号がそれ以上変わることはない．上位および下位データ・ストローブ信号（$\overline{UDS}$，$\overline{LDS}$）*は，S4の始めにアサートされる．ライト動作ではこれら2つの信号がデータ・バス上のデータの有効性を示しているので，真を示すストローブ信号としてこれらを使うことができる．

ウェイト状態

周辺はデータ転送アクノリッジ信号（$\overline{DTACK}$）をアサートしてデータ・スト

＊　68008では信号$\overline{DS}$.

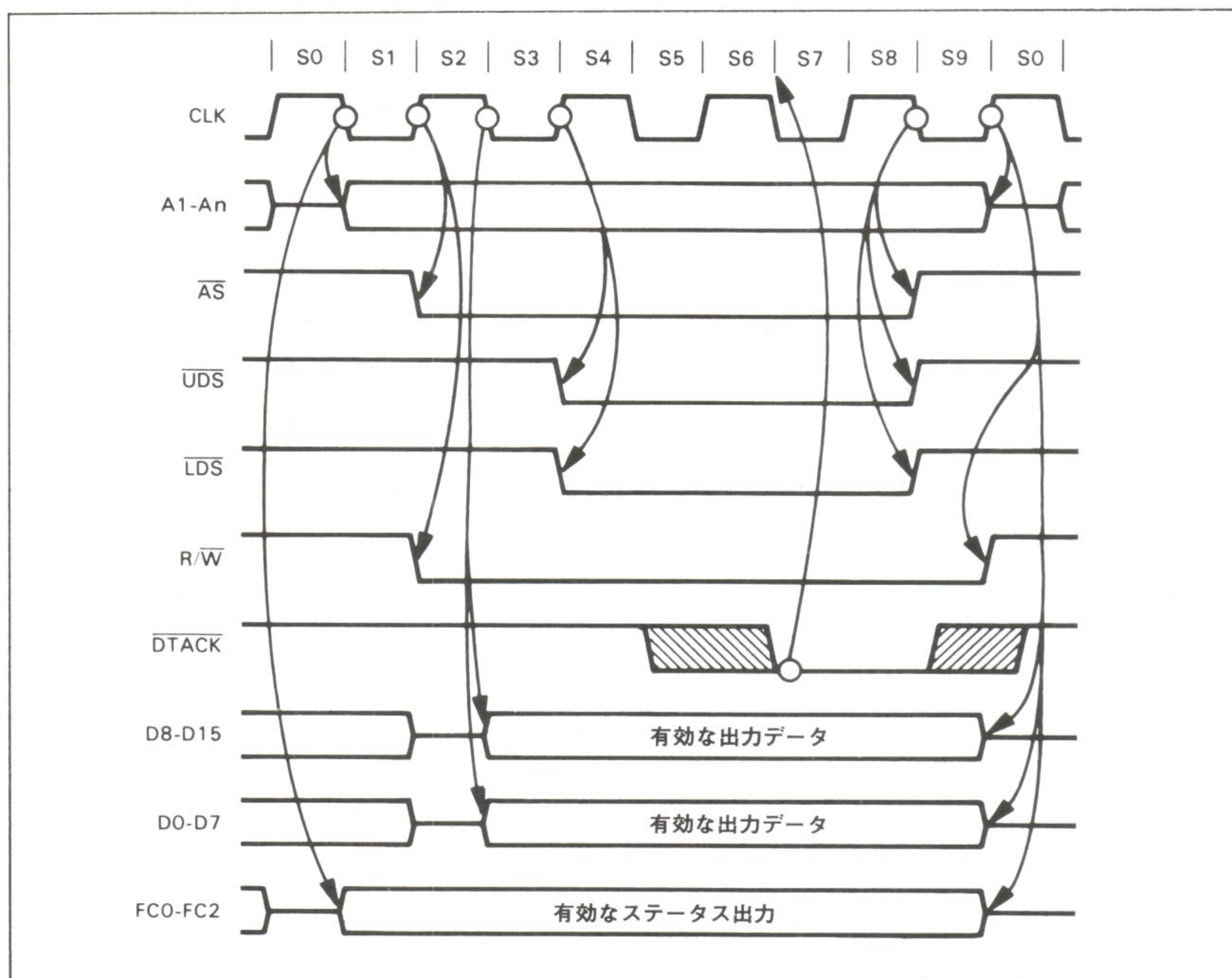



図6.4　ワード・ライトのタイミング（68000，68010，68012）

ローブに答えなければならない．S7の始めまでにそうしておけば，プロセッサは何の支障もなく，この状態を継続する．$\overline{\text{DTACK}}$ がS7の始めまでに真になっていなければ，68Kはライト・サイクル中にウェイト状態を自動的に挿入するが，このサイクル中への挿入が異なった時点*で起こることを除けば，この機能はリード・サイクルの場合と同様である．

68Kの場合は，ライト動作中は，データをD0-D15に出力している．アドレス・ストローブ（$\overline{\text{AS}}$）とデータ・ストローブ（$\overline{\text{LDS}}$，$\overline{\text{UDS}}$）は，S9の始めにネゲートされ，R/$\overline{\text{W}}$ 信号はS9の終わりでハイとセットされる．この時点でアドレス・バスとデータ・バスとファンクション・コードの出力はすべて高インピーダンス状態に戻り，システム・バスが他でも使えるようになる．周辺はアドレス・ストローブ信号やデータ・ストローブ信号のネゲートを検出した後に，$\overline{\text{DTACK}}$ 信号をネゲートしなければならない．これによって，次のバス・

＊　時点とはリード・サイクルではS5のとき，ライト・サイクルではS7のときである．

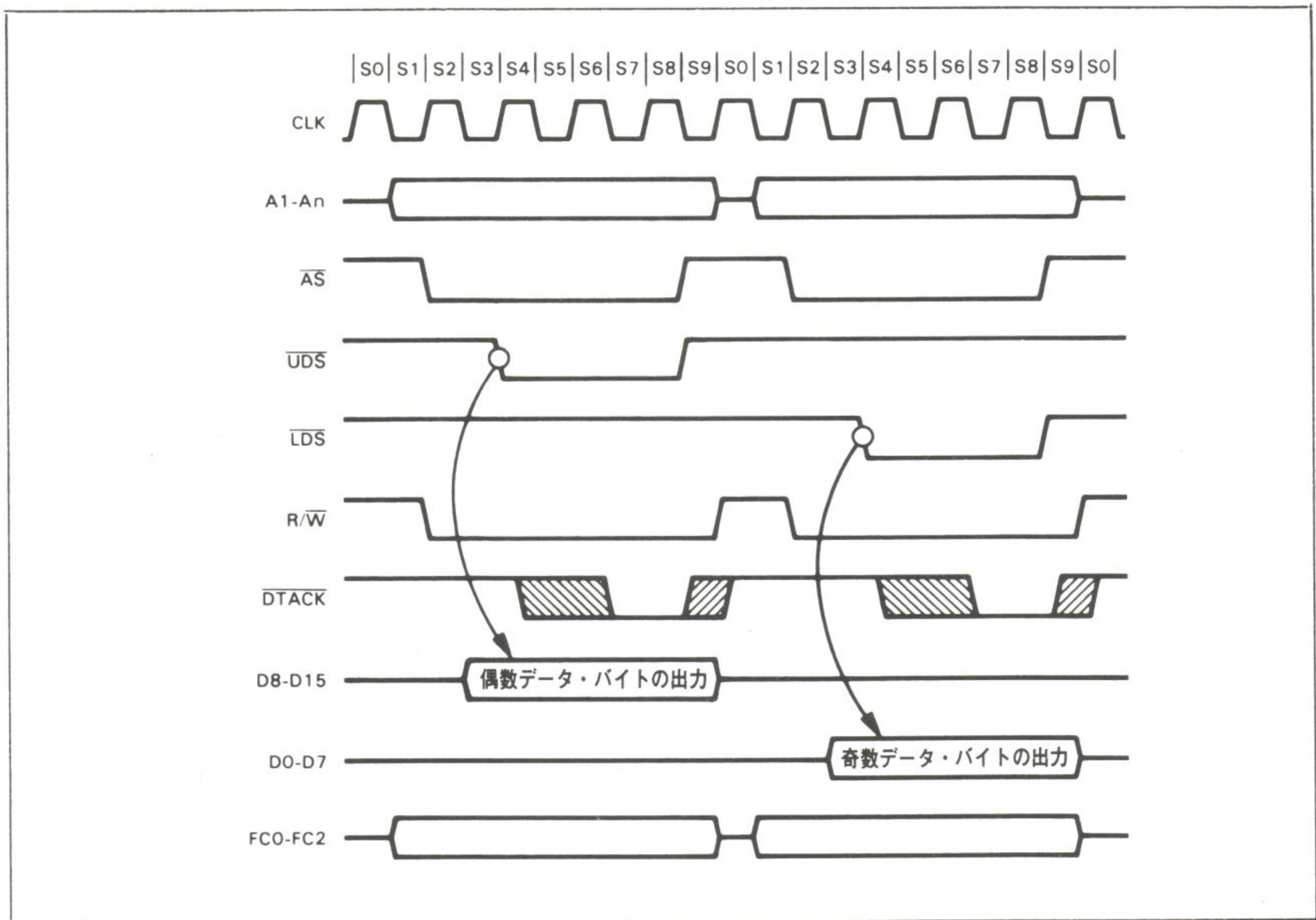



図6.5 バイト・ライトのタイミング（68000，68010，68012）

サイクルを支障なく確実に実行できることが保証される．

バイト・ライト・タイミング

バイト・ライトのタイミングを**図6.5**に示す．ワード・ライトのタイミングとの違いは，バイト書込みに際して，$\overline{\mathrm{UDS}}$ か $\overline{\mathrm{LDS}}$ のいずれかがアサートされることだけである．

68008でのライト・タイミング

68008において，バイト・ライトは，他のプロセッサと同様に機能するが，データ・ストローブ（$\overline{\mathrm{DS}}$）が1本だけであり，データ・バスが8本（D0-D7）しかない点が異なる．次ページの**図6.6**に68008のライト・サイクルを示す．

リード-モディファイ-ライト・タイミング

68Kのもっている**リード-モディファイ-ライト・サイクル**は，マイクロプロセッサでは珍しい機能である．68Kでは，TAS（テストセット）命令を実行する際のみこのサイクルを用いる．この命令は1バイトのデータを読み，その内容に応じてコンディション・コードをセットし，次にバイトの第7ビットをセットし，それをメモリ内に書き戻す．

TAS命令には，マルチプロセッサ・システムにおけるマイクロプロセッサ間の「安全な」通信を確保しようとする意図がある．リード-モディファイ-ライト・サイクルには，バスで接続している他の強力なマスタのバス要求でも割り込めない．したがって，TAS命令の実行中にこの命令でアクセスされたデータを他のバス・マスタでアクセスすることはできない．

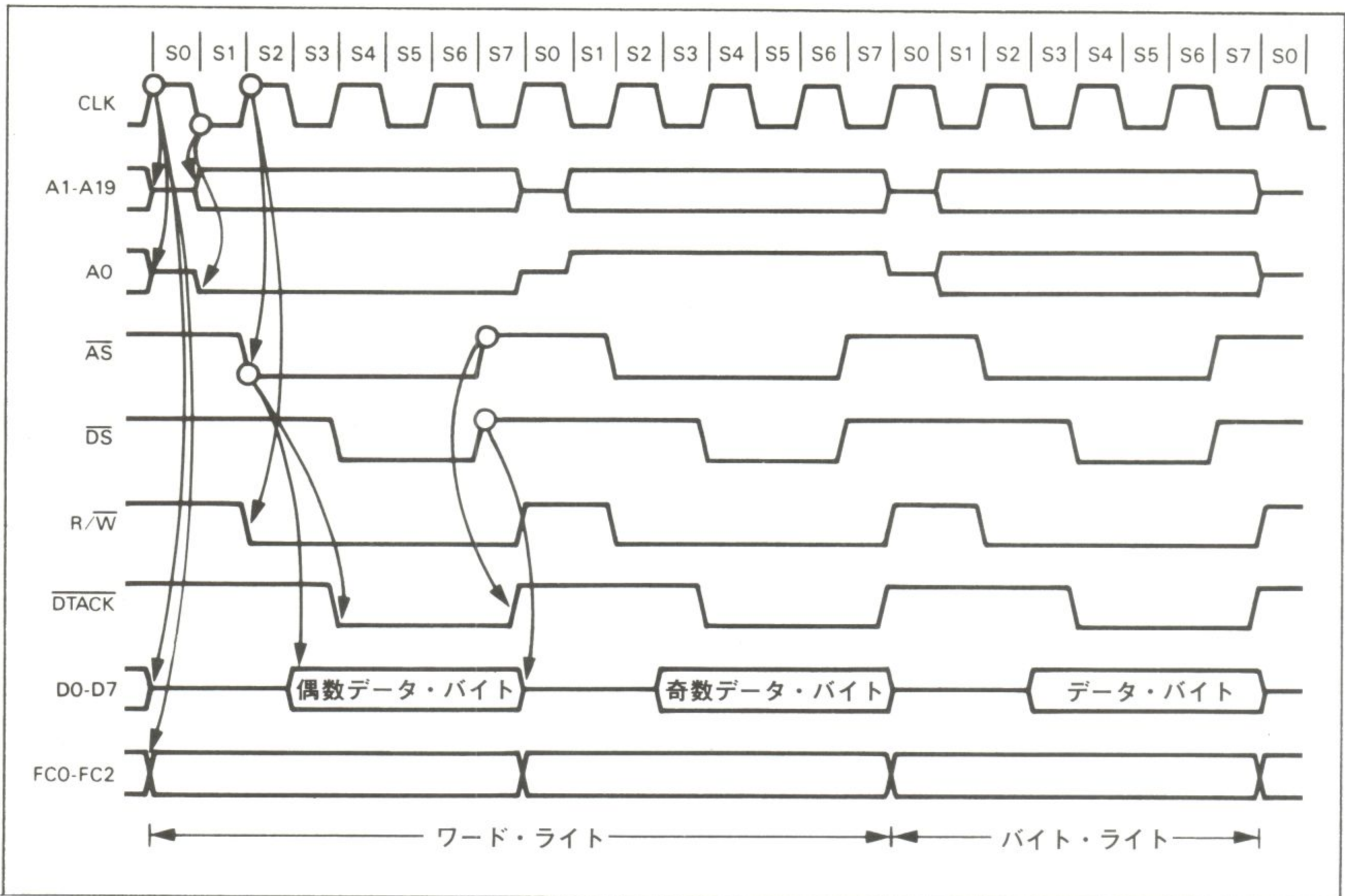



図6.6　ワードとバイトのライト・タイミング（68008）

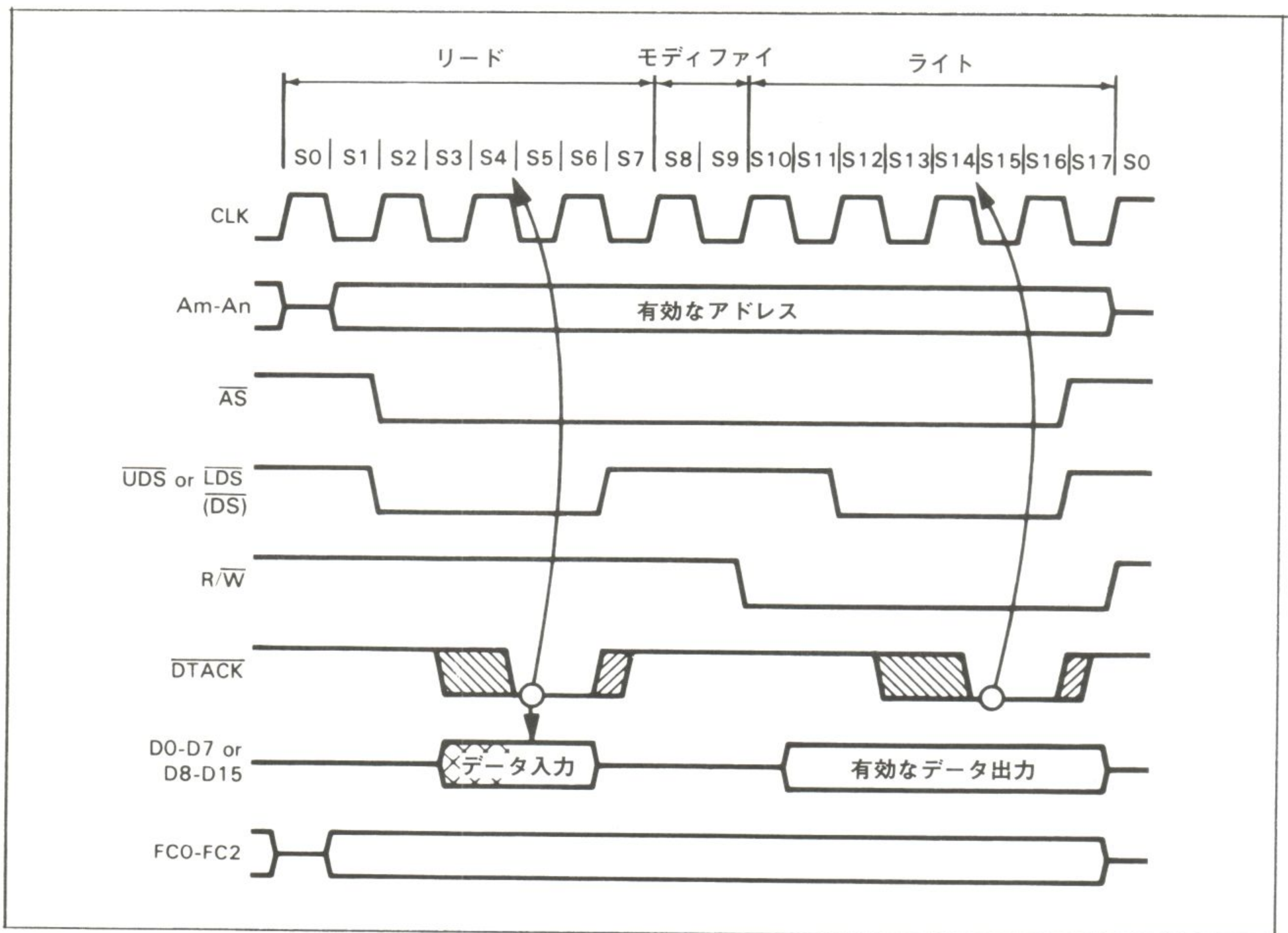



図6.7　リード-モディファイ-ライトのタイミング（68000-68012）

図6.7にリード-モディファイ-ライト・サイクルのタイミングを示す.

68020のタイミングとバスの動作

この章の最初に述べたように，68020におけるタイミングとバス動作は，動的バス幅設定とオペランドの不当整列という点で，前に述べたプロセッサとは異なっている．また，68020にはCPU空間に関する精妙な動作とプロセッサをシステム・バスに接続する新しい信号がある.

リードもライトもバス幅とアドレス整列に影響されるから，始めに動的バス幅設定とオペランドの不当整列について述べる．これらの概念が分かれば，後半におけるリード/ライトが分かりやすくなるであろう.

プロセッサは動的バス幅設定を用いて，1，2，4バイト幅の任意サイズをもつ周辺と1，2，3，4バイトの任意幅での転送をすることができる．この動的バス幅設定は，オペランドの不当整列を許容する上で重要な役割りを演ずる．今までの68Kプロセッサではワードおよびロングワード・オペランドを偶数アドレスから始める必要があったが，68020では偶数，奇数を問わずどのアドレスからでも任意幅のオペランドでアクセスできる.

これはオペランドの取出しだけに応用できることに注意されたい．効率をよくする都合上，68020の命令語は偶数アドレス境界から配置するように定められており，奇数アドレスで命令をアクセスしようとすると，アドレス・エラー・ベクタをアクセスして例外処理を始める.

使用できる制御ライン

3種のバス・ラインがあって，プロセッサのデータ・バスにおけるデータ転送を効率のよいものにしている．それは転送幅ライン（SIZ0とSIZ1），データ転送幅アクノリッジライン（$\overline{\text{DSACK0}}$と$\overline{\text{DSACK1}}$），2本の下位アドレス・ライン（A0とA1）である.

SIZ0とSIZ1は，プロセッサの転送しようとするバイトを示す．**表6.1**にバイト数の解釈を示した.

$\overline{\text{DSACK0}}$，$\overline{\text{DSACK1}}$は，周辺のデータ経路の幅を示す．プロセッサから転送の要求が出た後，転送サイクルの始めの時点で，そのアドレスに対応して，

表6.1 SIZ0/SIZ1のエンコード

SIZ0	SIZ1	バイト数
0	1	1
1	0	2
1	1	3
0	0	4

周辺は一度に扱えるバイト数をプロセッサに知らせる必要がある．**表6.2**に$\overline{\text{DSACK0}}$と$\overline{\text{DSACK1}}$がポート幅をどう解釈するかを示した．

表6.2　$\overline{\text{DSACK0}}/\overline{\text{DSACK1}}$のエンコード

$\overline{\text{DSACK0}}$	$\overline{\text{DSACK1}}$	意　　味
H	H	ポートがレディでない(ウェイト状態の挿入)
H	L	8ビット・ポート・レディ
L	H	16ビット・ポート・レディ
L	L	32ビット・ポート・レディ

(H：ハイ，L：ロー)

A1とA0はデータ転送の整列状態を示す．その状態によって，転送の境界がロングワードかワードかバイトになる．**表6.3**にA0，A1の可能な状態を示した．

表6.3　A0-A1のエンコード

A0	A1	ロングワード境界からのオフセット
0	0	＋0バイト（ロングワード，ワード境界）
0	1	＋1バイト
1	0	＋2バイト（ワード境界）
1	1	＋3バイト

転送されるバイトの整列状態と個数によって，プロセッサの用いる転送サイクルの個数は1から4になる．これを1バイト，2バイト，3バイト，4バイトと分ける．転送をするにあたって，プロセッサはポート幅に従い，データ・バスを一部または全部用いる．たとえば，奇数アドレスでロングワードを転送するには，そのアドレスに従って，1バイト/2バイト/1バイト，または1バイト/3バイトという形をとる必要がある．

バス幅と整列の複雑さを理解するには，A0/A1と$\overline{\text{DSACK0}}/\overline{\text{DSACK1}}$で示されるアドレスの整列状況とポート幅に対して，SIZ0/SIZ1で示される転送における残留バイト数を考えるのが最も便利である．転送バイトを区別するための記号を**図6.8**に示す．

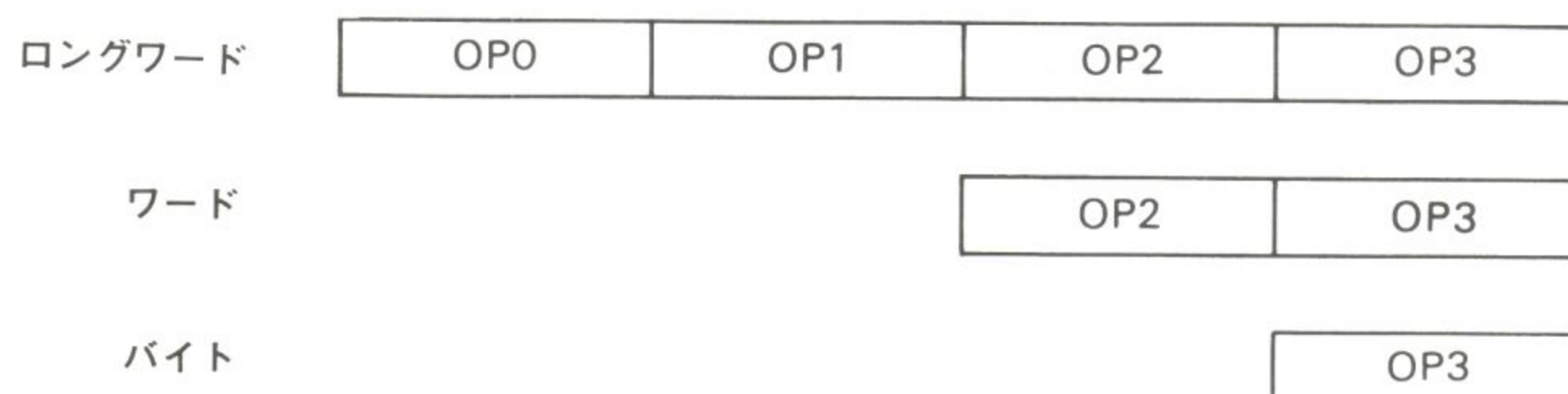



図6.8　バイト名に関する規約

表6.4 各ポートに対するデータ・バスの使用

転送幅	A1	A0	データ・バスの使用部			
			D31-D24	D23-D16	D15-D8	D7-D0
1 バイト (SIZ1/SIZ0=01)	0 0 1 1	0 1 0 1	BWL B BW B	— WL — W	— — L —	— — — L
2 バイト (SIZ1/SIZ0=10)	0 0 1 1	0 1 0 1	BWL B BW B	WL WL W W	— L L —	— — L L
3 バイト (SIZ1/SIZ0=11)	0 0 1 1	0 1 0 1	BWL B BW B	WL WL W W	L L L —	— L L L
4 バイト (SIZ1/SIZ0=00)	0 0 1 1	0 1 0 1	BWL B BW B	WL WL W W	L L L —	L L L L

(B=8ビット・ポート，W=16ビット・ポート，L=32ビット・ポート)

データ・バス・マルチプレクサ

こういった各種の状態でのデータ転送を容易にするため，プロセッサは，データ・バス・マルチプレクサを利用する．マルチプレクサによるデータ・バスの使い方は，転送幅とアドレス整列に関係し，転送されるバイトに応じて異なっている．**表6.4**に4種類の転送幅に対するデータ・バスのデータ位置をまとめた．

表6.5に示すように，データ・バスのどの部分でもデータをもてる．これは大切なことなので，特にここに書いておくこととした．ただし，外部ロジックには余計なデータを無視する方法がある．

表6.5 外部データ・バスとマルチプレクサ

転送幅	A1	A0	データ・バス上にアサートされるデータ			
			D31-D24	D23-D16	D15-D8	D7-D0
1 バイト(SIZ1/SIZ0=01)	X	X	OP3	OP3	OP3	OP3
2 バイト(SIZ1/SIZ0=10)	X X	0 1	OP2 OP2	OP3 OP2	OP2 OP3	OP3 OP2
3 バイト(SIZ1/SIZ0=11)	0 0 1 1	0 1 0 1	OP1 OP1 OP1 OP1	OP2 OP1 OP2 OP1	OP3 OP2 OP1 OP1	OP1 OP3 OP2 OP1
4 バイト(SIZ1/SIZ=00)	0 0 1 1	0 1 0 1	OP0 OP0 OP0 OP0	OP1 OP0 OP1 OP0	OP2 OP1 OP0 OP1	OP3 OP2 OP1 OP0

(X：何でもよい)

バス上のデータ転送は次の3ステップで処理される．

1. プロセッサがアドレスとデータ幅に関する情報をアサートし，周辺は1サイクルで全データを転送するものと，最初は仮定する．ライトの場合は，表6.4に示したように，データ・バスをロードする．リードの場合は，データ・バス上でデータを求めるが，それを知らせるため周辺でウェイトをかける．
2. 周辺はできる範囲でなるべく広くデータ・バスをロードするかまたは受け付けてから $\overline{\mathrm{DSACK0}}/\overline{\mathrm{DSACK1}}$ をそれに応じてセットする．
3. プロセッサは転送されたバイト数をSIZ0とSIZ1から減じ，A0とA1に加える．もしバイトが転送されないで残っていれば，このステップを繰り返す．

整列/データ幅に関する例

データ転送の例を示す．最初は32ビットの周辺をロングワード境界で用いるロングワード転送の例で，次のように1サイクルでやれる（A1/A0＝00）．

転　送	A1	A0	SIZ1	SIZ0	D31 - D24	D23 - D16	D15 - D8	D7 - D0
第1回	0	0	0	0	OP0	OP1	OP2	OP3

次は，32ビットの周辺を奇数アドレスで用いるロングワード転送の例である．データ転送は次のように2サイクルを要する．

転　送	A1	A0	SIZ1	SIZ0	D31 - D24	D23 - D16	D15 - D8	D7 - D0
第1回	0	1	0	0	—	OP0	OP1	OP2
第2回	0	0	0	1	OP3	—	—	—

次も32ビットの周辺を奇数アドレスで用いるロングワード転送の別の例である（A1/A0＝11）．データ転送は2サイクルを要する．

転　送	A1	A0	SIZ1	SIZ0	D31 - D24	D23 - D16	D15 - D8	D7 - D0
第1回	1	1	0	0	—	—	—	OP0
第2回	0	0	1	1	OP1	OP2	OP3	—

第4番目の例として，16ビットの周辺を偶数アドレスで用いるロングワード転送を示す（A1/A0＝10）．データ転送は2サイクルを要する．

転　送	A1	A0	SIZ1	SIZ0	D31 - D24	D23 - D16	D15 - D8	D7 - D0
第1回	1	0	0	0	OP0	OP1	—	—
第2回	0	0	1	0	OP2	OP3	—	—

次は8ビットの周辺をロングワード境界で用いるロングワード転送の例である（A1/A0＝00）．データ転送は4サイクルを要する．

転　送	A1	A0	SIZ1	SIZ0	D31 - D24	D23 - D16	D15 - D8	D7 - D0
第1回	0	0	0	0	OP0	—	—	—
第2回	0	1	1	1	OP1	—	—	—
第3回	1	0	1	0	OP2	—	—	—
第4回	1	1	0	1	OP3	—	—	—

第6番目の例に，16ビットの周辺を奇数アドレスで用いるワード転送を示す(A1/A0=01)．データ転送は2サイクルを要する.*

転　送	A1	A0	SIZ1	SIZ0	D31 - D24	D23 - D16	D15 - D8	D7 - D0
第1回	0	1	1	0	—	OP2	—	—
第2回	1	0	0	1	OP3	—	—	—

最後は32ビットの周辺を奇数アドレスで用いるワード転送の例である(A1/A0=01)．データ転送は1サイクルでできる．

転　送	A1	A0	SIZ1	SIZ0	D31 - D24	D23 - D16	D15 - D8	D7 - D0
第1回	0	1	1	0	—	OP2	OP3	—

これらの例からわかるように，表6.4を用いて考えれば，データ転送は非常に単純である．32ビットの周辺を用いる転送が最も効率的なことは明らかであろう．さらに，ロングワード転送はアドレスがロングワード境界となっているとき（A1/A0=00)，最も効率的である．

リード・サイクルのタイミング

データ幅と整列に関する知識を基として，リード・サイクルのタイミングを見ることが必要である．

1サイクルのロングワード・リード

32ビット・ポートがロングワードを読み取るにあたり，プロセッサは次の信号を使って読取りを始める．プロセッサは$R/\overline{W}$をリード（ハイ）とセットし，適当なファンクション・コードを出力し，所定のアドレスをアドレス・バスから出力し，転送幅信号SIZ0，SIZ1をそれぞれセットする．プロセッサはサイクル・スタート信号（$\overline{OCS}$と$\overline{ECS}$)，アドレス・ストローブ（$\overline{AS}$)，データ・ストローブ（$\overline{DS}$)，データ・バッファ・イネーブル・ライン（$\overline{DBEN}$）をアサートする．

周辺がアドレスおよびデータ・ストローブを感知すると，アドレス・バスの

---

* この例を後ほど不当整列データのリードとライトで用いる．

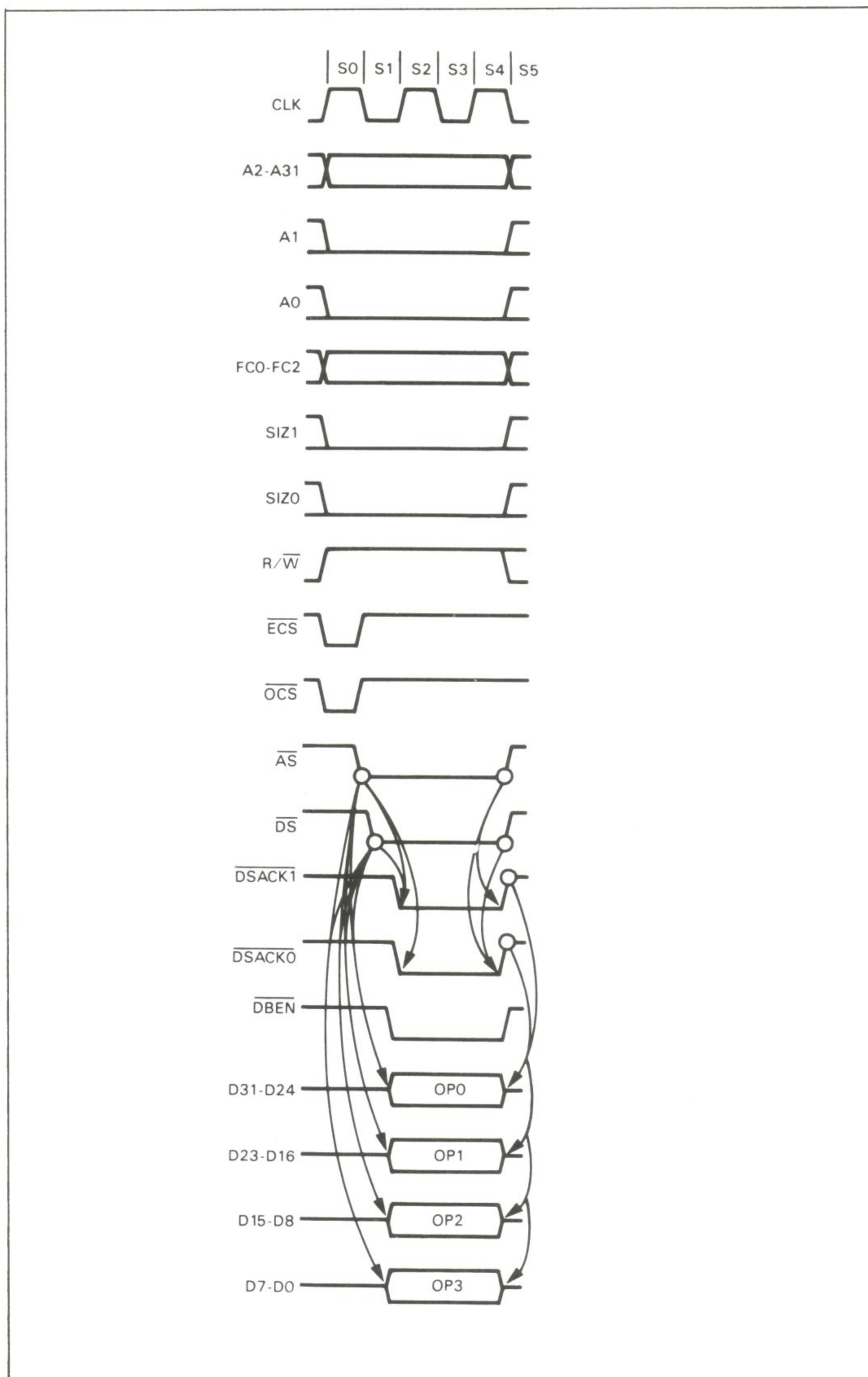



図6.9　ロングワード・リードのタイミング（32ビット・ポート）

アドレスをデコードし，データ・バスにそれぞれのデータを置き，$\overline{\text{DSACK0}}$と$\overline{\text{DSACK1}}$を両方アサートする．こうして4バイト全部がデータ・バスにのったことが示される．

プロセッサがS2の終わりまでに$\overline{\text{DSACK0}}/\overline{\text{DSACK1}}$を感知しなければ，自動的にウェイト状態をサイクル中に挿入する．もし感知すれば，データ・バスからデータをラッチし，$\overline{\text{DS}}$，$\overline{\text{AS}}$，$\overline{\text{DBEN}}$をネゲートするので，周辺はデータ・バスの使用を止め，信号$\overline{\text{DSACK0}}$と$\overline{\text{DSACK1}}$をともにネゲートする．こうして，プロセッサは次のサイクルに進む．**図6.9**にこの動作のタイミングを示した．

バイトとワードのリード

32ビット・ポートによるバイトとワードのリード機能も同様であるが，A0/A1およびSIZ0/SIZ1の状態が異なる．次ページの**図6.10**にそれを示した．

不当整列データのリード

前述したように，ロングワード境界にないデータのアクセスは，データ・バスの多重化と多重サイクルの使用可能性を含んでいた．表6.4に示したように，プロセッサはデータ・バス上に分割したデータをロードし待機している．多重サイクルのアクセスは，次にあげる2点を別として，単一サイクルのアクセスと同様である．

① プロセッサは$\overline{\text{OCS}}$信号を1度だけアサートし，$\overline{\text{ECS}}$信号を各フェッチの始めにアサートする．

② SIZ0/SIZ1とA1/A0は，残されたオペランド部のデータ幅とアドレスを示すため，各アクセス間に変わる．

奇数アドレスにおける16ビット・ポートのワード・リードのタイミングを69ページの**図6.11**に示す．これは前述の第6例（65ページ）に相当する．

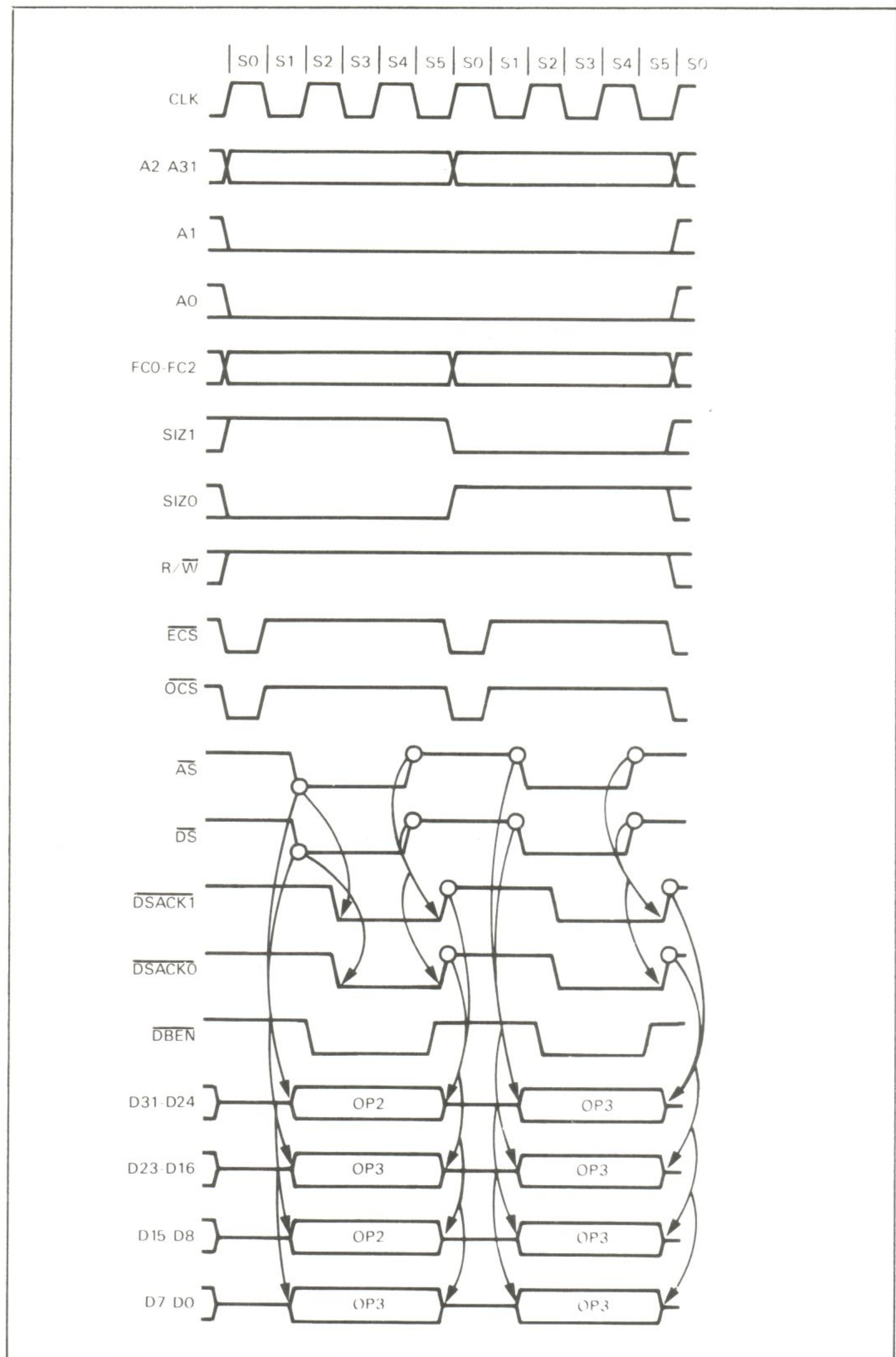



図6.10　ワード・リード，バイト・リードのタイミング（32ビット・ポート）

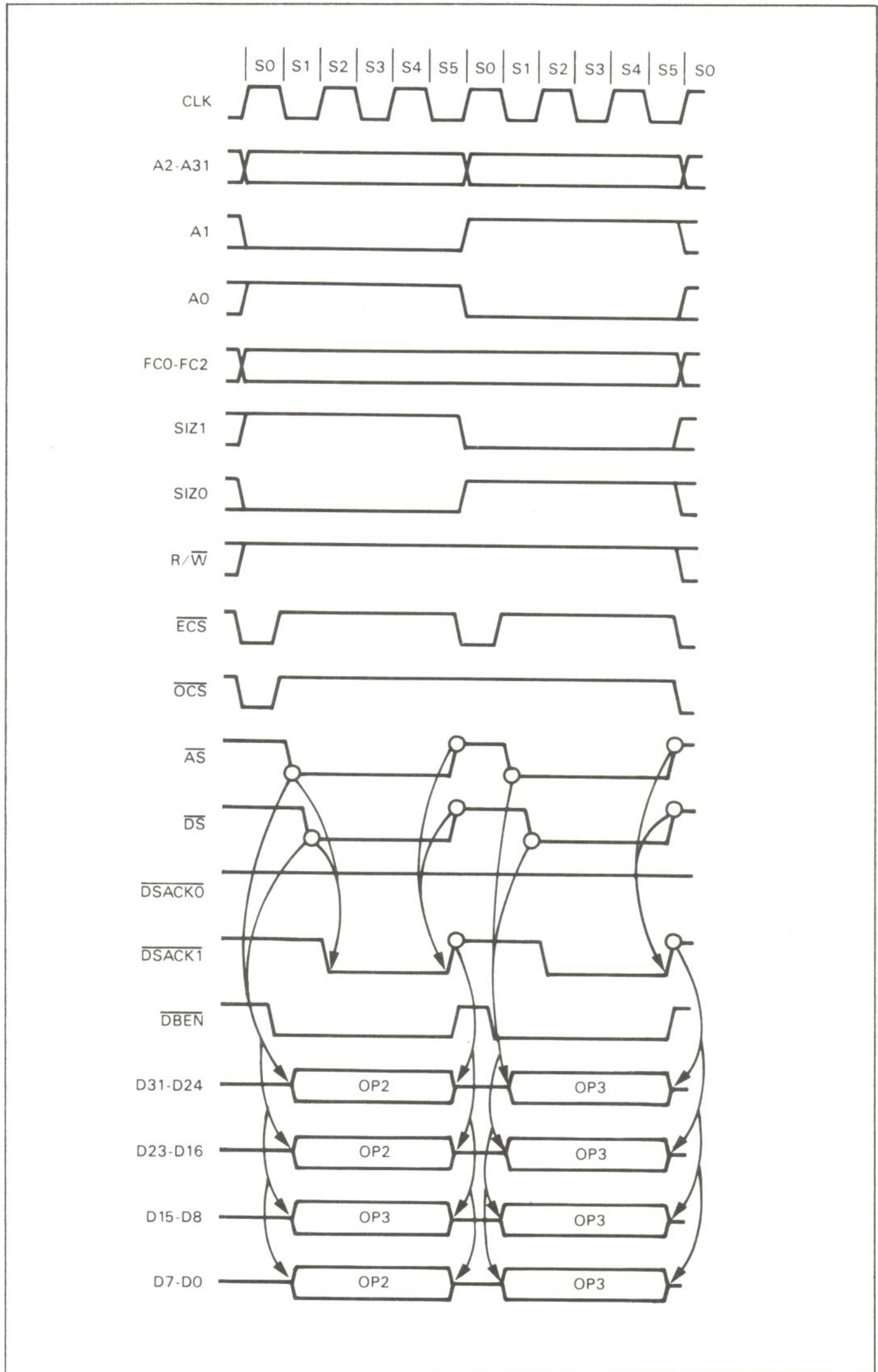



図6.11 不当整列ワード・リードのタイミング（16ビット・ポート）

ライト・サイクルのタイミング

1サイクルのロングワード・ライト

32ビット・ポートにロングワードを書き込むにあたり，プロセッサは次の信号を使って書込みを始める．

① プロセッサはR/$\overline{W}$をライト（ロー）とセットし，適当なファンクション・コードを出力し，アドレス・バスから所定のアドレスを出力し，転送幅信号SIZ0，SIZ1をそれぞれセットする．

② プロセッサはサイクル・スタート信号（$\overline{OCS}$と$\overline{ECS}$）とアドレス・ストローブ（$\overline{AS}$）をアサートし，データをデータ・バスに出力し，データ・バッファ・イネーブル・ライン（$\overline{DBEN}$）をアサートする．

周辺がアドレス・ストローブを感知したら，$\overline{DSACK0}$/$\overline{DSACK1}$によってデータ幅をプロセッサに示す．周辺が応答しなければ，$\overline{DSACK}$のいずれか一方または両方がアサートされるか，外部ロジックがバス・エラー（$\overline{BERR}$）を出すまで，プロセッサが自動的にウェイト状態を挿入する．プロセッサが$\overline{DSACK0}$/$\overline{DSACK1}$信号を受け取ると，データ・ストローブ（$\overline{DS}$）をアサートし，周辺がデ--タをラッチしなければならない．

こうしてプロセッサは$\overline{DS}$，$\overline{AS}$，$\overline{DBEN}$をネゲートし，データ・バスからデータを除く．周辺がこれを感知したとき，$\overline{DSACK0}$と$\overline{DSACK1}$をともにネゲートし，プロセッサは次のサイクルを始める．**図6.12**にロングワード・ライトのタイミングを示した．

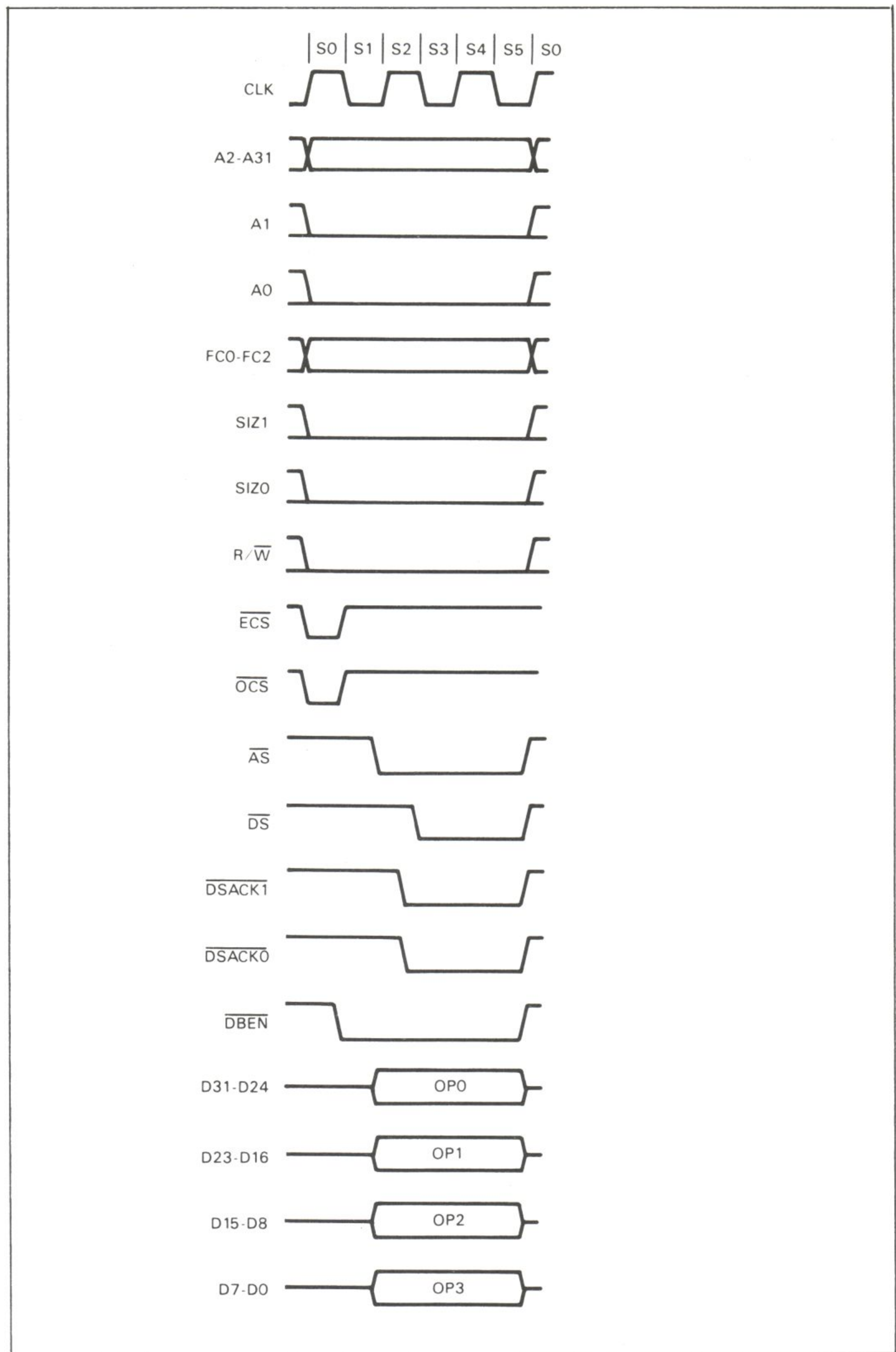



図6.12 ロングワード・ライトのタイミング（32ビット・ポート）

バイトとワードのライト

バイトとワードの32ビット・ポートによるライトも同様であるが，上位アドレス・ライン（A0とA1）とデータ幅信号（SIZ0とSIZ1）の状態が異なる．**図6.13**に32ビット・ポートによるロングワード境界からのワードおよびバイトについてライトのタイミングを示した．

不当整列データのライト

前述したように，ロングワード境界にないデータのアクセスは，データ・バスの多重化と多重サイクルの使用可能性を含んでいた．表6.4に示したように，プロセッサはデータ・バス上に分割したデータをロードして待っている．多重サイクルのアクセスは，次の点を別として，単一サイクルのアクセスと同様である．

① プロセッサは$\overline{\mathrm{OCS}}$信号を1度だけアサートし，$\overline{\mathrm{ECS}}$信号を各フェッチの始めにアサートする．

② SIZ0/SIZ1/A0は，残されたオペランド部のデータ幅とアドレスを示すため，各アクセス間に変わる．

奇数アドレスにおける16ビット・ポートのワード・ライトのタイミングを74ページの**図6.14**に示す．これは前の第6例（65ページ）に相当する．

リード-モディファイ-ライト・タイミング

以前の68Kと同じく，68020にも不可分な**リード-モディファイ-ライト・サイクル**がある．その目的は，アクセスの開始から完了までにおいて，オペランドが不意に変わることはないと保証することである．マルチプロセッサ・システムでは，制御プロセッサがオペランドを調べ（変更し）ている間に，可能なバス・マスタがバス要求$\overline{\mathrm{BR}}$によってそこへ割り込めるので，これは大変有効である．

以前の68Kプロセッサはリード-モディファイ-ライト・サイクルを行うための命令として，TAS（テストとセット）命令しかもっていなかったが，68020ではそこへ比較とスワップ（CAS，CAS2）命令を2つ加えた．

TAS命令はバイト・オペランドで機能するが，CAS（CAS2）命令はどんなオペランドでも動作する*．これらはアドレスやポート幅に従って4バイトまでのデータで動作するので，動作完了までに多くのバス・サイクルが必要である．命令実行中はバスがプロセッサを支配しているから，バス選択応答に影響する．理念的には，CAS命令やCAS2命令でアクセスされるセマフォーは，すべてロングワード境界にある．TAS命令はバイト・データをアクセスするだけなので，ここでは使えない．

75ページの**図6.15**に32ビット・ポートに対するCAS命令のタイミングを示した．

---

* バイト，ワード，ロングワードで動作する．

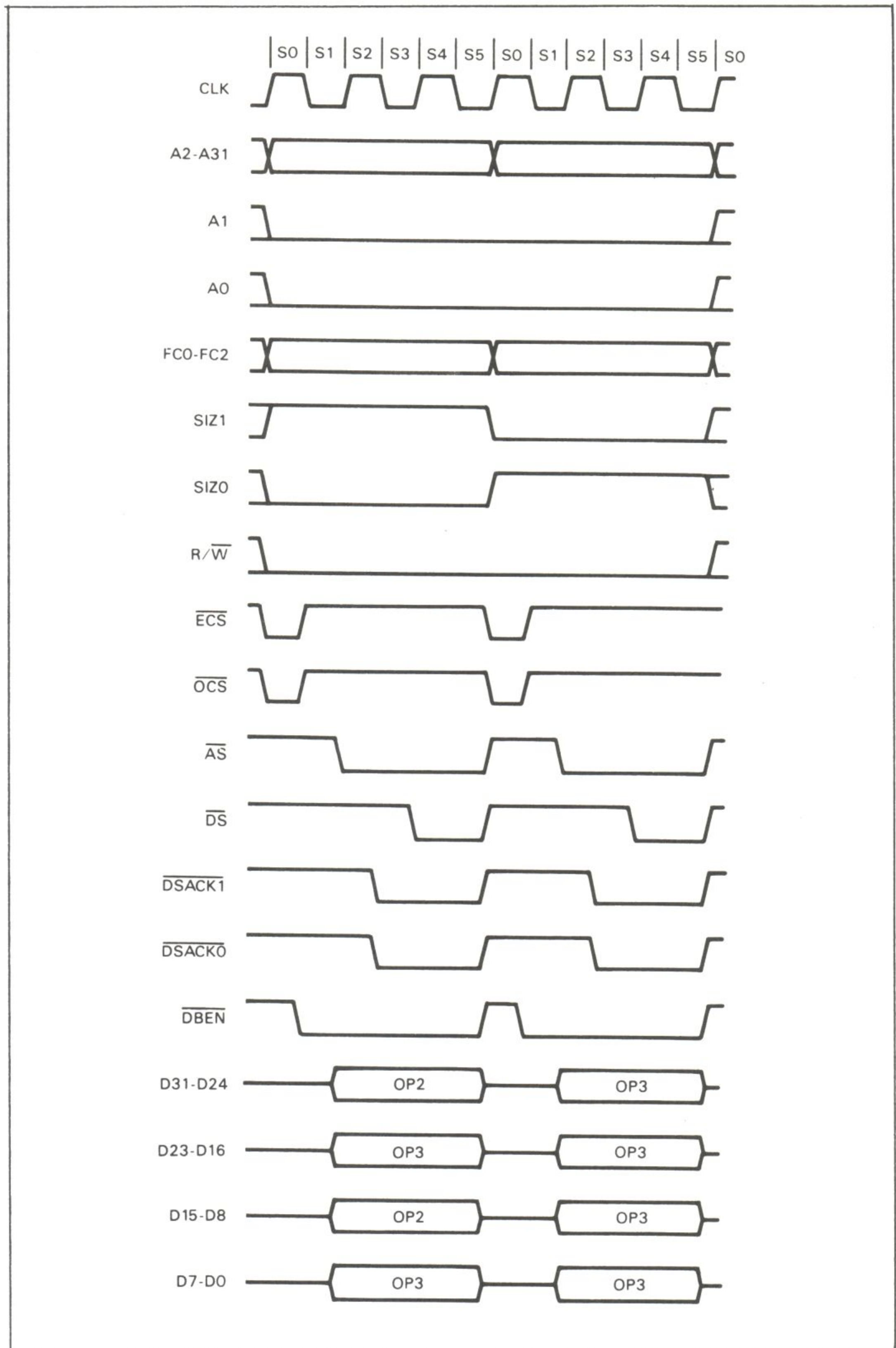



図6.13 ワード・ライト，バイト・ライトのタイミング（32ビット・ポート）

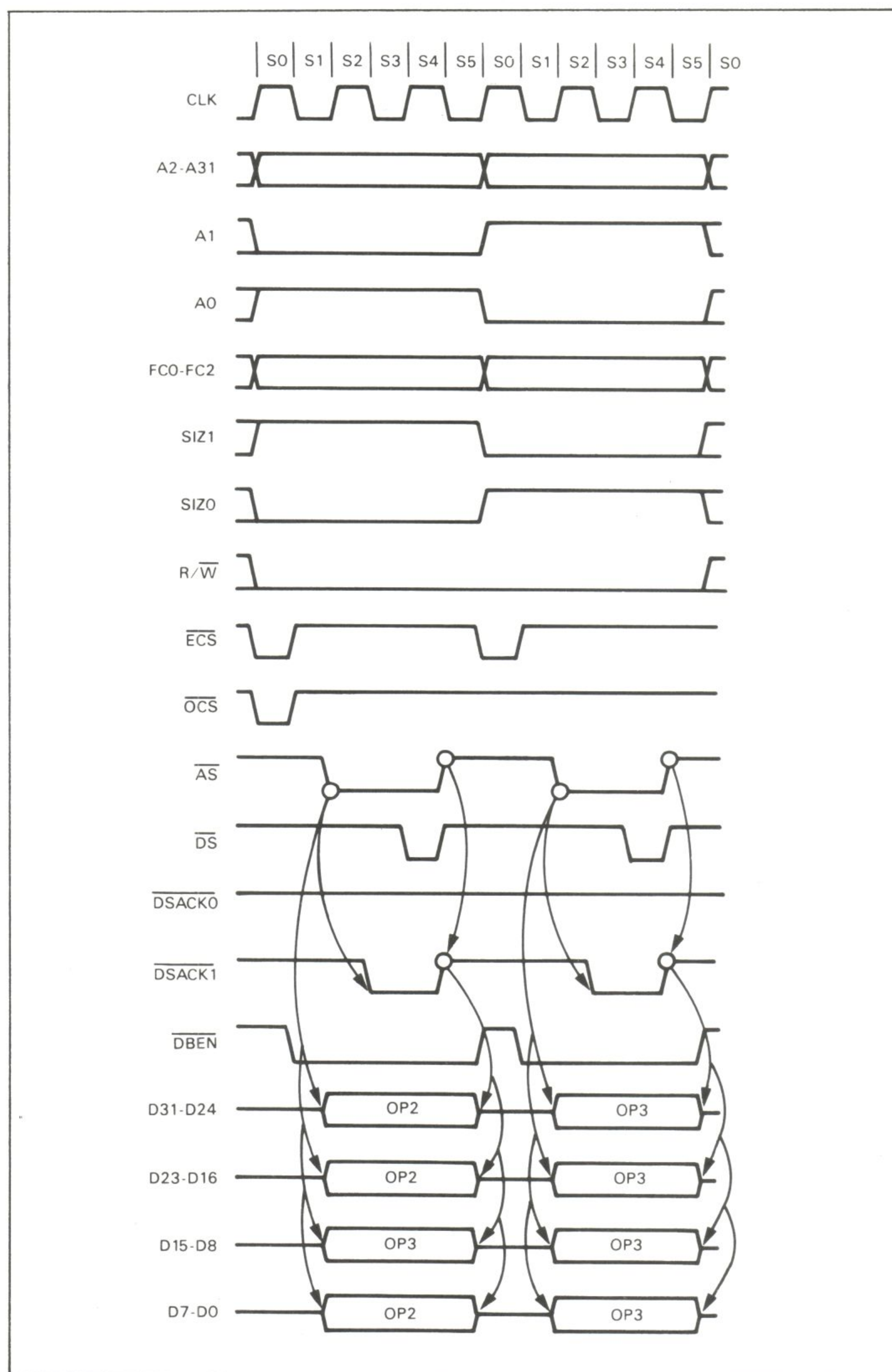



図6.14　不当整列ワード・ライトのタイミング（16ビット・ポート）

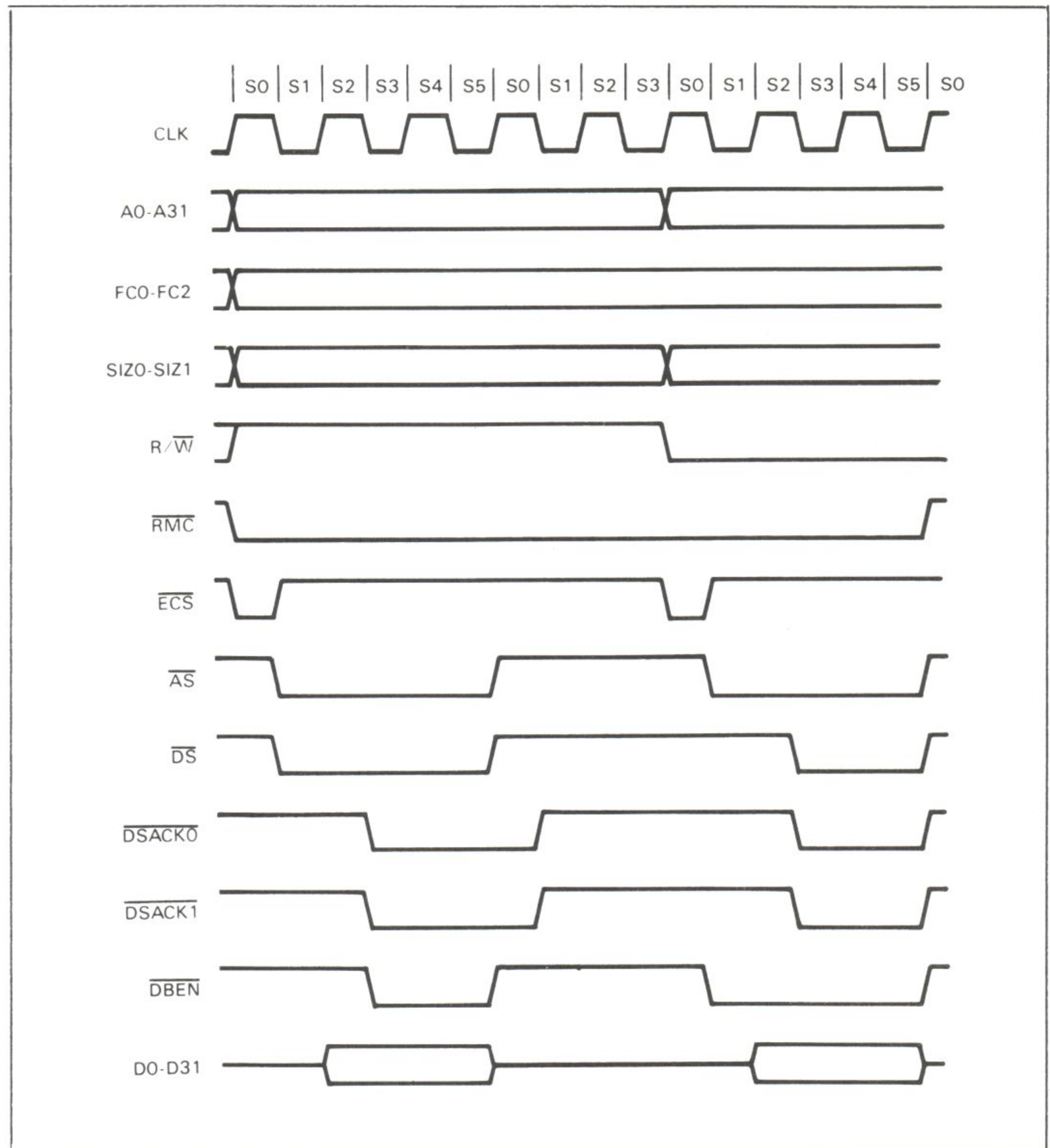



図6.15 リード-モディファイ-ライトのタイミング（CAS 命令，32ビット・ポート）

68Kファミリに共通な特性

今までは68020とそれ以前の68Kを分けて，リード/ライトのタイミングをみてきたが，これからは68K全体に共通なバス動作を取り扱うこととする．

リセット動作

ハードウェア・リセット

68Kプロセッサには非同期のリセット入力がある．68000，68008，68010，68012では，$\overline{\text{RESET}}$信号と$\overline{\text{HALT}}$信号を，最低100ミリ秒間アサートすると，プロセッサがリセットされる．68020でも同様だが，ここでは$\overline{\text{RESET}}$信号だけをアサートすればよい．信号をネゲートすると，プロセッサは次のように動作する．

1. 68Kはベクタ・テーブルにある最初のロングワード（オフセット0）を読み取り，システム・スタック・ポインタ（SSP）にロードする．また次のロングワード（オフセット4）をプログラム・カウンタ（PC）にロードする．
2. 68Kはステータス・レジスタ（SR）にある割込みマスクをすべて1にセットする．これはノン・マスカブル割込み(non-maskable interrupt)信号だけがプロセッサに割り込めるという意味である．ステータス・レジスタのスーパバイザ・ビット(S)がセットされて，スーパバイザ・モードで動作するようになり，トレース・ビット(T)はクリアされる．68010，68012，68020ではベクタ・ベース・レジスタ（VBR）がアドレス00000000にセットされる．68020では，命令キャッシュが使用禁止になり，ステータス・レジスタのマスタ・ビット(M)がクリアされる．
3. 68Kはプログラム・カウンタにロードされた新しいアドレスからプログラムの実行を始める．

汎用のデータおよびアドレス・レジスタの値はシステム・リセットによってすべて不定となる．

ソフトウェア・リセット

リセット信号は双方向性である．68Kがリセット命令を実行すると，124クロック・サイクル（68020では512クロック・サイクル）だけ$\overline{\text{RESET}}$ラインがアサートされる．この命令はプロセッサの内部状態には少しも影響せず，レジスタの内容も変わらない．この「ソフトウェア・リセット」により，$\overline{\text{RESET}}$バス・ラインと接続されたすべての周辺装置をリセットする命令がプロセッサから発せられる．

ホールト状態

強制ホールト

68Kは$\overline{\mathrm{HALT}}$ラインを外部からアサートして，「ホールト」状態にすることができる．現在のバス・サイクルの完了後，アドレス・バス，データ・バス，ファンクション・コードは高インピーダンス状態になり，ホールト信号がネゲートされるまでプロセッサは実行をストップする．プロセッサは外部ロジックに対するホールトの肯定応答を出さない．

プロセッサがホールト状態にあると命令は実行しないが，バス・アービトレーション回路は機能している．しかし68Kはバスを使っていないので，外部バス要求はすぐに使用を許可される．バス・アービトレーションについては本章で後ほど論ずる．

シングル・ステップ・モードと$\overline{\mathrm{HALT}}$

多くの命令の実行では，命令やオペランドをフェッチするため，複数個のバス・サイクルが必要である．プロセッサはバス・サイクルの完了後に$\overline{\mathrm{HALT}}$入力に応答するので，２命令の中間または単一命令の途中でホールト・シーケンスが起こり得る．したがって，動作をシングル・ステップ・モードで実行するように$\overline{\mathrm{HALT}}$入力を用いることができる.* プロセッサはその状態を示さないから，これは主としてハードウェア・デバッグを行ったことになる．命令単位のシングル・ステップ・モードの実行は，ステータス・レジスタにあるトレース・ビット（T）を用いたトレース機能により行うことができる．次章でソフトウェア・トレースについて述べよう．

HALT出力信号

$\overline{\mathrm{HALT}}$信号は双方向性である．68Kで，（バス・エラー例外ベクタをフェッチしようとしてバス・エラーを起こす）ダブル・バス・フォールトが発生すると，プロセッサはホールト状態に入り，$\overline{\mathrm{HALT}}$信号をアサートして重大な障害が生じたことをプロセッサに知らせる．プロセッサを再スタートさせる唯一の方法は，ハードウェア・リセット動作によることである．

ストップ状態

ストップ状態は，プロセッサが本質的には何もしないという点ではホールト状態に似ている．特権命令であるSTOPを実行すると，ストップ命令に続くイミディエイト・データをステータス・レジスタにロードし，プログラム・カウンタに次の命令のアドレスをロードしてから，ストップ状態に入る．

ホールト状態と異なり，プロセッサは状態を示す特別な信号を出力しない．ホールト状態は障害を意味し，リセットによってのみ再スタートするが，割込

＊　１命令ずつの実行という意味ではなく，バス・サイクル単位という意味である．

みの生成のように，ストップ状態は例外条件が生ずると終了する．

バス・サイクルの再実行

すでに述べたように，$\overline{\mathrm{BERR}}$ のアサートで示されるシステム・バス・エラーに対し，68K は2つの方法で応答できる．第7章で論ずるが，1つは例外処理を行うことであり，他はバス・エラーを起こしたバス・サイクルを再実行してみることである．$\overline{\mathrm{BERR}}$ だけがアサートされていれば，例外処理が始まる．しかし，$\overline{\mathrm{BERR}}$ 信号が $\overline{\mathrm{HALT}}$ 信号を伴っていれば，68K はこれをバス・サイクルを再実行するための要求と認める．

この場合 68K は実行中であったサイクルを完了させるように動作し，ホールト状態に入る．アドレス・バス，データ・バス，ファンクション・コードはすべて高インピーダンス状態になり，$\overline{\mathrm{BERR}}$ と $\overline{\mathrm{HALT}}$ がともにネゲートされるまでプロセッサはホールト状態を続ける．68K が単独の $\overline{\mathrm{BERR}}$ 信号をソフトウェアで処理すべき別のバス・エラーと解釈しないようにするため，$\overline{\mathrm{HALT}}$ がネゲートされる少なくとも1クロック・サイクル前に，$\overline{\mathrm{BERR}}$ をネゲートしなければならない．

$\overline{\mathrm{HALT}}$ がネゲートされると，68K は再実行の要求を受けたとき実行中であったサイクルを繰り返す．すなわち，直前のバス・サイクルで用いたのと同じアドレス，データ，ファンクション・コードに関する情報を使って，再実行を行う．

再実行の成功

期待した時間内に $\overline{\mathrm{DTACK}}$ を受けると，再実行サイクルがうまく完了したことが分かる．もちろん，いつもこうなるとは限らない．バス・サイクルを再実行しようとしてまたバス・エラーを起こす場合もある．外部ロジックは，$\overline{\mathrm{BERR}}$ と $\overline{\mathrm{HALT}}$ を組み合わせて用いることにより，サイクルの無制限再実行を要求することができる．しかし，$\overline{\mathrm{BERR}}$ を単独でアサートして，バス・エラーの処理にソフトウェアによる例外処理を用いれば，バス・エラーが2回続く場合，「ダブル・バス・フォールト」は重大なエラーとして扱われ，68K は自動的にホールト状態に入り，リセットするまでその状態を続ける．

68K がリード - モディファイ - ライト・サイクルを実行しているとき，バス・エラーを起こすと，そのサイクルは再実行できない．これは，リード - モディファイ - ライト・サイクルは TAS（テストとセット）命令を実行している間だけ行われているからである．この命令の本質は，実行サイクル全体の完全性を要求したもので，バス・サイクルのどこでも再実行されれば，完全性は消えてしまう．外部ロジックがリード - モディファイ - ライト・サイクルの再実行を要求すると，68K はその代わりにバス・エラーの例外処理ルーチンを実行する．

バス・アービトレーション・ロジック

68K が備えたバス・アービトレーション・ロジックは，単純である．68K は外部デバイスによるバス・アクセス要求に優先順位をつけない．プロセッサがその時点でバスを使っていなければ，プロセッサは要求を行ったどんなデバイスにも常に，バス・アクセスを許可するので，プロセッサの優先順位がシステムで最も低いものと考える．

したがって，68K では命令と命令の間および単一命令のバス・サイクルの間で，他のデバイスがバスを用いてよい．アービトレーション・ロジックが内蔵されていないので，システム・バスの要求に優先順位をつけるために，外部バス・アービトレーション・ロジックをつけた複雑なシステムもあり，優先順位の高いデバイスが低いデバイスに使用権を奪われないようになっている．

バス・アービトレーション・ロジックに関しては，**バス・リクエスト**（$\overline{\mathrm{BR}}$），**バス・グラント**（$\overline{\mathrm{BG}}$），**バス・グラント・アクノリッジ**（$\overline{\mathrm{BGACK}}$）という 3 つの信号がある．68K が競合しないシステム・バスを使っていれば，入力信号 $\overline{\mathrm{BR}}$ と $\overline{\mathrm{BGACK}}$ はインアクティブで出力信号 $\overline{\mathrm{BG}}$ はネゲートされている．

バス・アービトレーションのタイミング

図6.16に，68K が行っているバス・アービトレーションのタイミングを示す．外部デバイスが入力 $\overline{\mathrm{BR}}$ をアサートすると，バス・アービトレーションが開始される．68K がバス・リクエストを受け取ると，1 クロック・ピリオド遅れて $\overline{\mathrm{BG}}$ をアサートしてそれに応える．ただし，68K がバス・サイクルの初期状態にあって，$\overline{\mathrm{AS}}$ をアサートしていない場合はすぐには応えられない．この場合には，68K は $\overline{\mathrm{AS}}$ がアサートされた後に，$\overline{\mathrm{BG}}$ をアサートするまで 1 クロック・ピリオド待っている．この場合の応答時間は最大で 3 クロック・ピリオドになる．

バス・グラントの決定

バス・グラント信号だけがアサートされた時点では，要求したデバイスがバスを使えるかどうかは分からない．現在のバス・サイクルが完了するまでは，68K がバスを使っている．したがって，バスを要求しているデバイスは，バスが実際に利用できる時点を定めるために，他の信号をいくつも監視しなければならない．

最初に，外部デバイスは，$\overline{\mathrm{AS}}$ がネゲートされるまで待つ必要がある．これは 68K の現在のバス・サイクルが完了したことを示す．また，現在の 68K のサイクルがバスを使っていないことを知るため，$\overline{\mathrm{DTACK}}$（68020 では $\overline{\mathrm{DSACK0}}$/$\overline{\mathrm{DSACK1}}$）信号がネゲートされるまで待つ必要がある．$\overline{\mathrm{DTACK}}$ や $\overline{\mathrm{DSACK0}}$/$\overline{\mathrm{DSACK1}}$ を監視する必要のないシステムもある．これは $\overline{\mathrm{AS}}$ がネゲートされたとき，システムのタイミングが，どの外部デバイスもバスを使っていないことを，いつも保証できるような場合である．

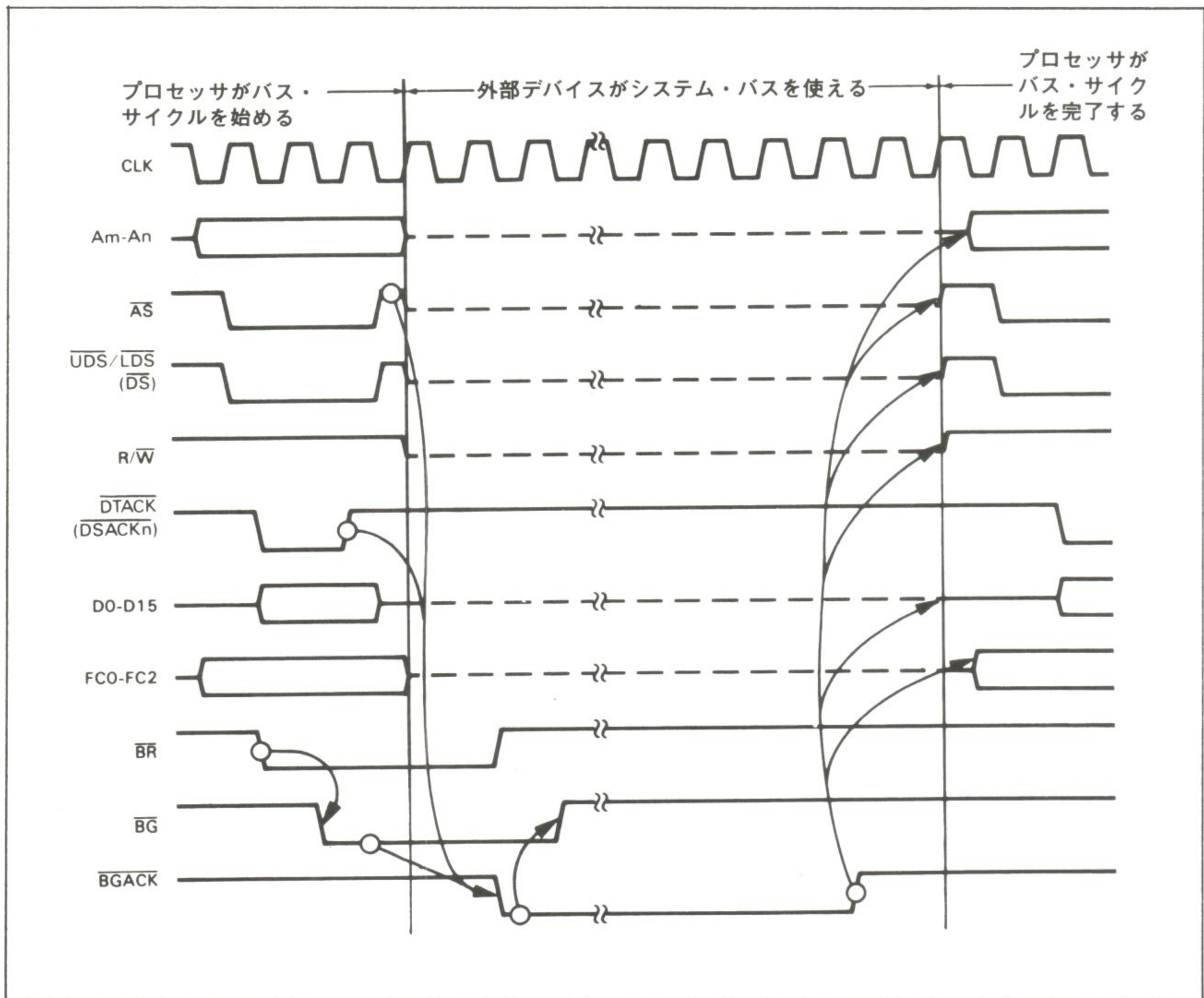



図6.16　バス・アービトレーションのタイミング

最後に，要求しているデバイスは，$\overline{\text{BGACK}}$ 信号の状態をチェックしなければならない．この信号がアサートされていれば，システムの他のデバイスがすでにバスの使用を認められていて，それがまだ完了していないことを示している．逆に $\overline{\text{BGACK}}$ が偽であれば，現在のサイクルの終わりでシステム・バスが利用できる．

このような信号条件がすべて成立した後，バスを要求しているデバイスは，$\overline{\text{BGACK}}$ をアサートしなければならない．これによって，要求しているデバイスがバスを制御できるようになったことがプロセッサに分かる．図6.16では，68K が $\overline{\text{BGACK}}$ 信号を待たずにバスの制御を放棄していることに注意する．68K は進行中であったバス・サイクルが完了すると，アドレス・バス，データ・バス，ファンクション・コード，$\overline{\text{AS}}$，$\overline{\text{UDS}}$ と $\overline{\text{LDS}}$ （68008 と 68020 では $\overline{\text{DS}}$），R/$\overline{\text{W}}$ がすべてすぐに高インピーダンス状態になる．

バスを必要とする間に，バスを使っているデバイスは，$\overline{\text{BGACK}}$ 信号をアサ

ートしなければならない．外部デバイスがバスの制御を保持している間は，外部ロジックは$\overline{\text{BGACK}}$を監視してバスの競合を予防する．この時点では$\overline{\text{BR}}$と$\overline{\text{BG}}$の状態は重要でない．しかし，不正なバス要求を避けるため，バスを用いるデバイスは，$\overline{\text{BGACK}}$をネゲートする前に$\overline{\text{BR}}$をネゲートすべきである．

バスの回復

68Kは$\overline{\text{BGACK}}$がネゲートされるまで，出力ラインを高インピーダンス状態に保つ．この時点で，プロセッサは自由になり，別のバス・サイクルが始められる．この時点で別のバス要求が保留されていれば，68Kはすぐそのバス要求に従い，どんなバス・サイクルも実行しない．

第7章

例外処理

例外処理は，マイクロプロセッサで特殊な事態が生じたとき実行されるものである．この特殊な事態としては，**アドレス・エラー，バス・フォールト，特権違反**（ユーザ・モードで特権命令を実行しようとしたこと）および**0による除算**がある．これらはエラーによるものであるが，エラーによらない特殊な事態もある.* たとえば，周辺の割込み，ハードウェア・リセット，プログラムによる割込みもすべて例外処理を引き起こす．

動作モード

例外処理システムを述べる前に，例外処理に影響する68Kの動作モードを説明しよう．前述のように，68Kはスーパバイザ・モードかユーザ・モードで動作する．68Kを$\overline{\text{RESET}}$入力によりリセットすると，68Kはスーパバイザ・モードで動作を始め，プロセッサが次の命令のいずれかを実行するまで，このモ

---

* このような特殊事態を例外という．例外に伴って特定動作を起こすが，それを例外処理という．

ードを変えない．

① RTE ——例外からのリターン

② MOVE——ステータス・レジスタへの転送

③ ANDI ——ステータス・レジスタとの直接数値による論理積

④ EORI ——ステータス・レジスタとの直接数値による排他的論理和

これらの命令によって，動作が自動的にユーザ・モードに移行するのではなく，単にステータス・レジスタのＳビットの状態が変更されるにすぎない．しかしＳビットがリセットされると，68K はユーザ・モードで動作し始める．

68K が一度ユーザ・モードで動作すると，それをスーパバイザ・モードへ戻せるのは例外だけである．例外処理が始まると，プロセッサはスーパバイザ・モードに戻り，例外処理が完了すると RTE（例外からのリターン）命令を実行してユーザ・モードになる．

例外の型

例外はいろいろな状態で発生するが，次の２つに大別できる．

① ある命令を実行した結果，または，内部で検出されたエラーによる内部生成例外．

② バス・エラー，リセット，割込み要求などによる外部生成例外．

内部生成例外

内部生成例外はさらに**内部エラー，命令トラップ，トレース機能**の３種類に分かれる．内部エラーは内部的に検出されるもの*1 で，次の３つにより例外処理が始まる．

① **アドレス・エラー**——どんな命令も偶数アドレス境界になければならない．さらに 68020 を除いて，ワード・データやロングワード・データも偶数アドレス境界になければならない．不当整列されたデータや命令をアクセスしようとすると，例外処理が発生する．

② **特権違反**——前述のように，ある種の命令はスーパバイザ・モードでしか使えない．これらの特権命令はリソース（資源）をアクセスできるが，ユーザ・プログラムでは実行できず，例外処理を引き起こす．

③ **不当命令**と**未実装命令**——どの命令も 16 ビット長なので，命令として使われていないビット・パターンがたくさんある．もしプログラムが定義してない命令（不当命令）を実行しようとすると，例外処理が始まる．この中で，ビット 15-12 が 1010（A ライン）または 1111（F ライン）の命令*2 は不当命

＊1　ユーザ・プログラムの実行中にそこで生ずるエラー．

＊2　命令が＄A，＄F で始まる命令．1010，1111 は２進法による表現である．

令でなく，未実装命令と定められており，これに対しては不当命令に対するものと異なった例外処理が始まる．

この他68010，68012，68020では，＄4848から＄484Fというパターンをもった命令コードが「**ブレークポイント**」という特殊な不当命令になっている．後ほどこの点に触れよう．

命令トラップは，プログラムにおける命令実行により生ずる例外である．ここではオペレーティング・システムでよく用いる標準的なTRAP命令がある．これはアクセスをスーパバイザ・モードのリソースに限定して，ユーザ・モードのプログラムとするものである．他の命令としては，CHK, CHK 2 , cpTRAPcc，TRAPcc，TRAPV，DIVS，DIVUがあり，算術オーバフローやゼロによる除算などの条件を検出すると例外処理が始まる．

内部生成例外の第3のタイプは，68Kがトレース・モードで動作するときに起こる．ステータス・レジスタのTビット*をセットすると，各命令の実行後に例外処理が起こる．トレース機能により，各命令または指定した命令の実行後にプログラムをストップさせ，結果を解析することができるので，プログラムのデバッグには有効である．

外部生成例外

外部生成例外も3種類ある．それを次に示す．

① 外部ロジックにより$\overline{\text{BERR}}$がアサートされて生ずる**バス・エラー**に基づくもの．

② 外部ロジックにより$\overline{\text{RESET}}$がアサートされて生ずる**リセット**に基づくもの．

③ 外部ロジックが割込み要求ライン（$\overline{\text{IPL 0}}$-$\overline{\text{IPL 2}}$）をアサートして生ずる**割込み要求**に基づくもの．

例外の優先順位

例外処理の型によって優先順位が定まっている．リセット，バス・エラー，アドレス・エラーに対するものが最も優先順位が高く，バス・サイクル中であっても，これらの例外はどれも実行中の命令を直ちに打ち切ってしまう．トレース，割込み要求，不当命令と未実装命令，特権違反という例外がこれに続き，実行中の命令が完了してから例外処理を始める．割込み処理には割込み要求ラインの組み合わせに関係した割込み順位構造が追加されている．後ほどこれを論ずる．

＊ 68020でTに相当するのはT 1である．

表7.1　例外の優先順位

グループと優先度	特　　徴
0.0　リセット	すべての処理を打ち切る（それまでのプログラムなどは捨てられる）.
1.0　アドレス・エラー 1.1　バス・エラー	処理を一時中止する（それまでのプログラムなどはセーブする）.
2.0　BKPT，CALLM，CHK，CHK 2，cpにおける事中命令，* cpプロトコル違反，cpTRAPcc，0による除算，RTE，RTM，TRAP，TRAPV	例外処理は命令の通常の実行に含まれる.
3.0　不当命令，Fライン，Aライン，特権違反，cp事前命令	例外処理はこの命令を実行する前に始まる.
4.0　cp事後命令	実行中の命令や直前の例外処理が完了してから例外処理が始まる.
4.1　トレース 4.2　割込み	

* mid-instructionの訳. pre-instructionとpost-instructionのpre，postを事前，事後としたので，midを事中とした.

優先順位の最も低い例外はトラップ型の命令によるものである．これはその正常な実行の一部として，例外処理を始める．どの命令トラップも同時に2つ起こることはないから，優先順位はすべて等しい．

表7.1に,例外の型とその優先順位をまとめ，また例外処理の始まる時点も示した．

例外ベクタ・テーブル

68Kの例外処理シーケンスの中心となるのがベクタ・テーブルで，1024バイトのメモリを要する．このテーブルは,68000から68008では0000から03FFまでのアドレスを占めるが，他のプロセッサではベクタ・ベース・レジスタ(VBR)を使って，テーブルの先頭番地をユーザが定める．ハードウェアによるリセットでは，VBRは0になるので，テーブルは0000から始まる．

テーブルには4バイトのベクタが256個あり*，最初の8バイトは別として，各ベクタは32ビット・アドレスを示し，例外処理シーケンスの一部として，プ

* したがって例外ベクタの番号は0から255までとなる．

表7.2 例外ベクタの割当て

ベクタ番号	オフセット	割　当　て
0	000	リセット：割込みスタック・ポインタの初期値
1	004	リセット：プログラム・カウンタの初期値
2	008	バス・エラー
3	00C	アドレス・エラー
4	010	不当命令
5	014	0による除算
6	018	CHK，CHK 2 命令
7	01C	cpTRAPcc，TRAPcc，TRAPV命令
8	020	特権違反
9	024	トレース
10	028	Aライン・エミュレータ
11	02C	Fライン・エミュレータ
12	030	保留
13	034	コプロセッサ特権違反
14	038	フォーマット・エラー
15	03C	未初期化割込み
16-23	040-05C	保留
24	060	スプリアス割込み
25	064	オートベクタ（レベル 1）
26	068	オートベクタ（レベル 2）
27	06C	オートベクタ（レベル 3）
28	070	オートベクタ（レベル 4）
29	074	オートベクタ（レベル 5）
30	078	オートベクタ（レベル 6）
31	07C	オートベクタ（レベル 7）
32-47	080-0BC	トラップ＃0-15
48	0C0	浮動小数点コプロセッサ（FPCP）　不当状態による分岐またはセット
49	0C4	浮動小数点コプロセッサ（FPCP）　不正確な結果
50	0C8	浮動小数点コプロセッサ（FPCP）　0による除算
51	0CC	浮動小数点コプロセッサ（FPCP）　アンダフロー
52	0D0	浮動小数点コプロセッサ（FPCP）　オペランド・エラー
53	0D4	浮動小数点コプロセッサ（FPCP）　オーバフロー
54	0D8	浮動小数点コプロセッサ（FPCP）　有効値を表現しない（NAN）
55	0DC	保留
56	0E0	ページ・メモリ管理ユニット（PMMU）　構成（Configuration）
57	0E4	ページ・メモリ管理ユニット（PMMU）　違反命令
58	0E8	ページ・メモリ管理ユニット（PMMU）　アクセス・レベル
59-63	0EC-0FC	保留
64-255	100-3FC	ユーザ定義ベクタ

ログラム・カウンタPCにロードされる．最初の2ベクタはリセットに使われ，システム・スタック・ポインタとプログラム・カウンタに当てるものとして予約されている．このリセット・ベクタを除けば，どのベクタもスーパバイザ・データ空間を示すものとしてフェッチされる（**表7.2**）．

表からわかるように，ベクタ・テーブルの多数の項目が，前に述べた明確な

例外のために使われている．一部のベクタは後発の68Kファミリだけに関係しており，以前のモデルでは「保留」されていた．最初の64個のベクタは既定義または保留となっているが，残りの192個のベクタは，外部割込み要求のような一般の用途に利用できる．

スタック・フレーム

プロセッサと実行する例外の型によって，0ワードから44ワードにいたるデータがスーパバイザ・スタックに押し込まれる．これは例外処理シーケンスの一部である．このスタックに押し込んだデータをまとめて,「スタック・フレーム」という.**図7.1**に例外で用いるスタック・フレームの一般的なフォーマットを示した．

図7.1における「フォーマット」の作り方に注意して欲しい．68000，68012，68020にはスタック・フレームを定義する**フォーマット・フィールド**がいくつもある．68000と68008のスタック・フレームについてはそれほどきちんとしておらず，フォーマット・フィールドといったものはない．**表7.3**はスタック・フレーム・フォーマットの定義を要約したものである．

68000/68008のスタック・フレーム

トレース，TRAP，不当命令や未実装命令，特権違反，割込み要求に対する68000/68008のスタック・フレームの作り方を，**図7.2**に示した．またバス・エラーとアドレス・エラーに対するスタック・フレームの作り方を，**図7.3**に示した．

フォーマット$0のスタック・フレーム

図7.4にフォーマット＄0のスタック・フレームを示す．68010，68012，68020ではこのフォーマットを，割込み，フォーマット・エラー，TRAP命令，Aライン・トラップ，Fライン・トラップ，特権違反，コプロセッサの事前命令に関する各例外で用いる．

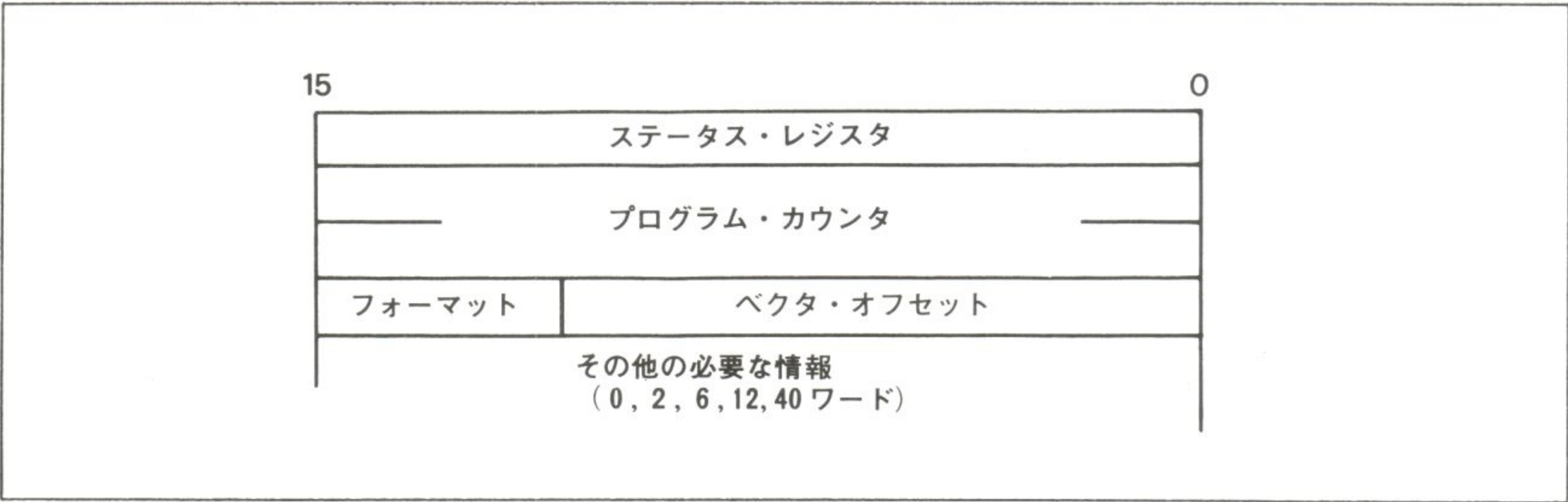



図7.1　スタック・フレームの一般的なフォーマット

表7.3　スタック・フレーム・フォーマットの定義

フォーマット	型
0000	ショートフォーマット（4ワード）
0001	スローアウェイ（4ワード）
0010	命令例外（6ワード）
0011-0111	保留
1000	68010-68012バス・フォールト（29ワード）
1001	コプロセッサ事中命令（10ワード）
1010	68020ショートバス・フォールト（16ワード）
1011	68020ロングバス・フォールト（44ワード）
1100-1111	保留

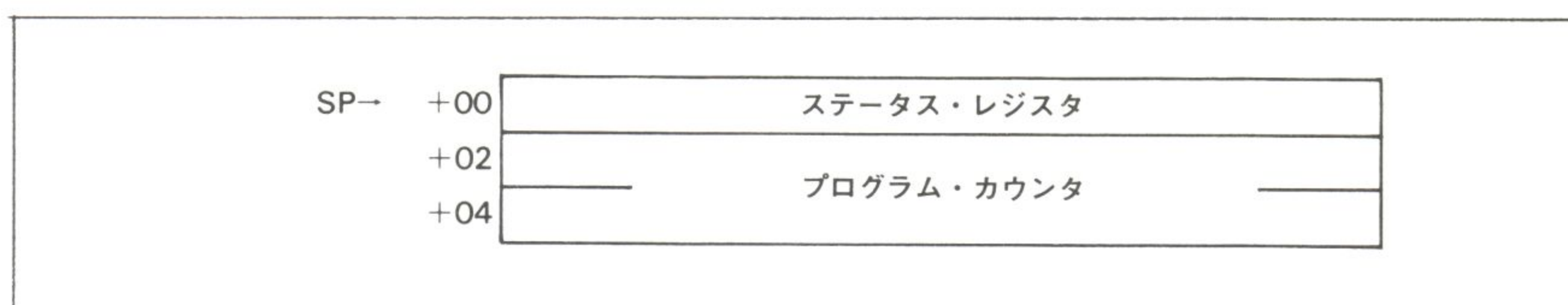



図7.2　68000/68008のショートスタック・フレーム

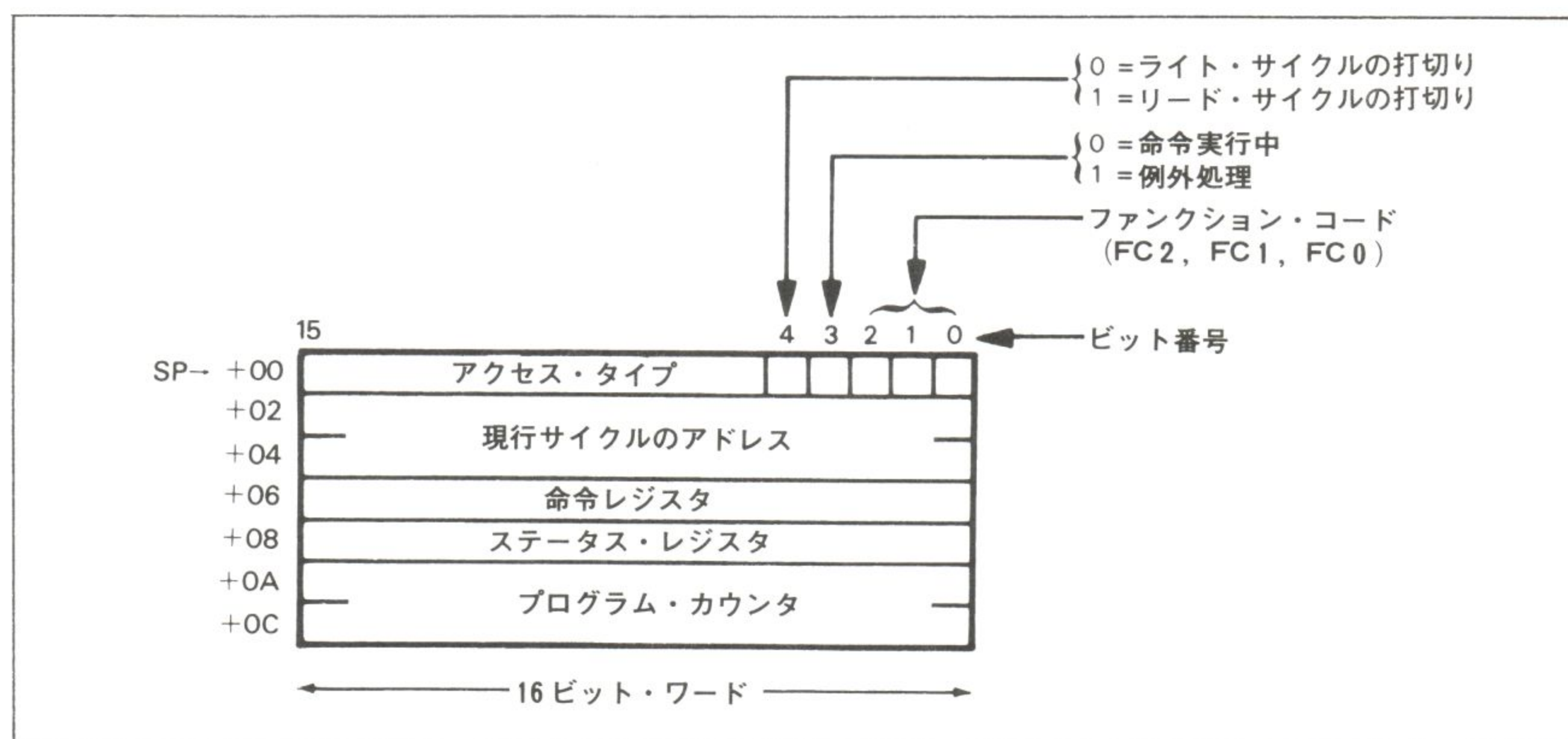



図7.3　68000/68008のバス・エラーとアドレス・エラーに対するスタック・フレーム

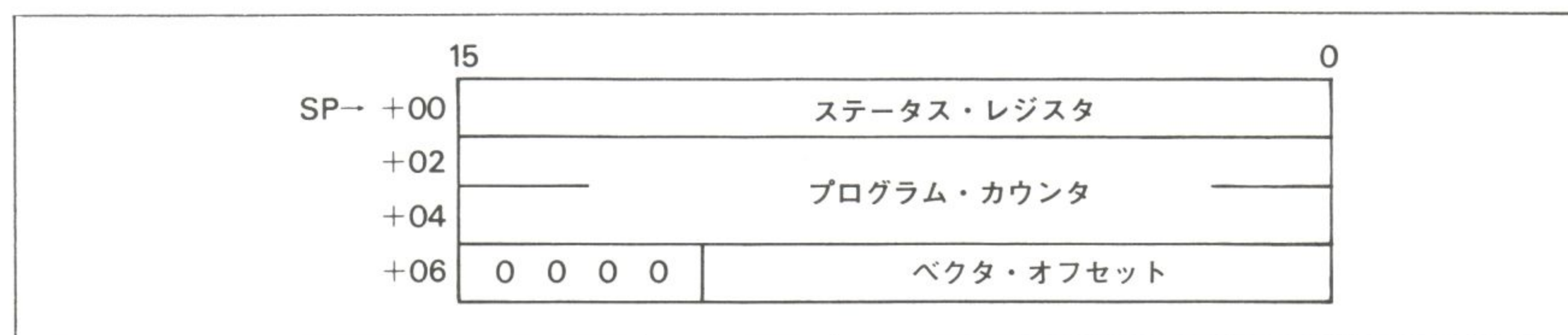



図7.4　フォーマット＄0のスタック・フレーム（4ワード）

フォーマット$1のスタック・フレーム

図7.5にフォーマット＄１のスタック・フレームを示す．これを「スローアウェイ」・スタック・フレームともいい，68020におけるステータス・レジスタのマスタ・ビットがセットされている間に割込みが発生したとき用いる．

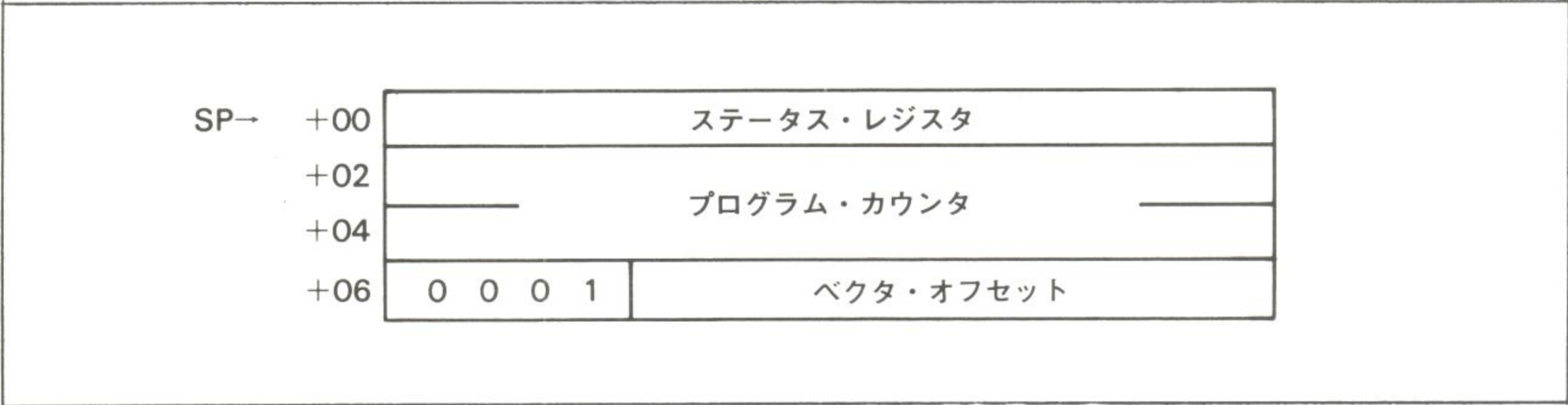



図7.5　フォーマット＄１のスタック・フレーム（４ワード）

フォーマット$2のスタック・フレーム

図7.6にフォーマット＄２のスタック・フレームを示す．これも68020で用いる．これはコプロセッサの事後命令，CHK，CHK２，cpTRAPcc，TRAPcc，TRAPV，トレース，０による除算に関する例外で使われている．

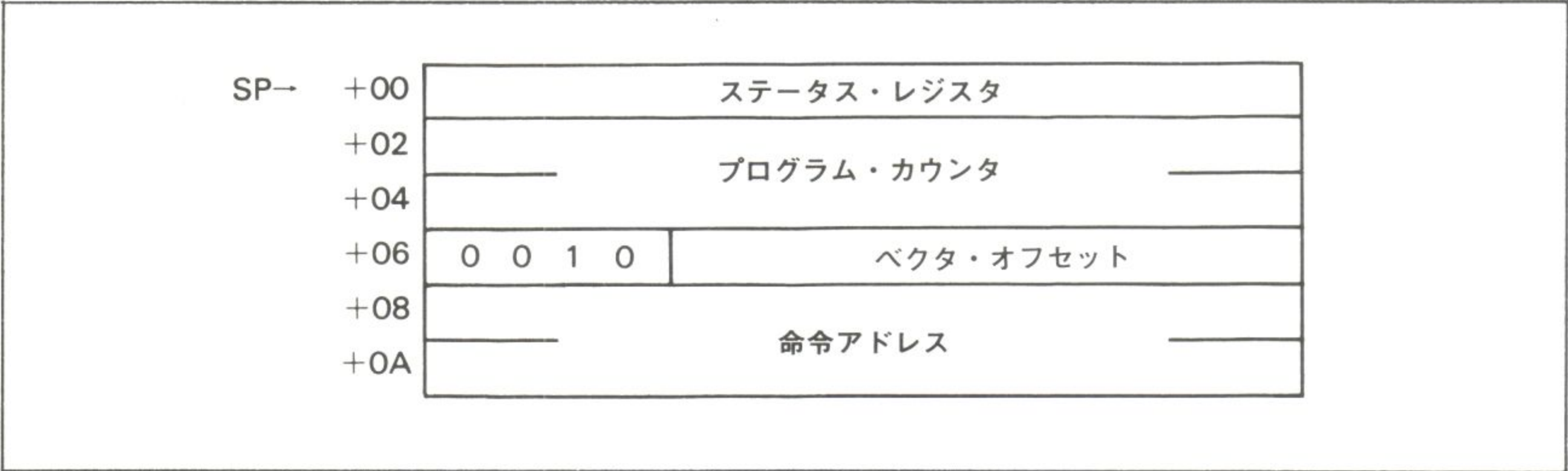



図7.6　フォーマット＄２のスタック・フレーム（６ワード）

フォーマット$8のスタック・フレーム

図7.7にフォーマット＄８のスタック・フレームを示す．68010と68012は，これをバス・エラー，アドレス・エラーに関する例外で用いる．

オフセット	内容
SP→ +00	ステータス・レジスタ
+02	プログラム・カウンタ
+04	
+06	1 0 0 0 / ベクタ・オフセット
+08	ステータス・ワード
+0A	フォールト・アドレス
+0C	
+0E	保留
+10	データ出力バッファ
+12	保留
+14	データ入力バッファ
+16	保留
+18	命令入力バッファ
+1A	内部情報（16ワード）
.	.
+38	.

図7.7　フォーマット＄８のスタック・フレーム(29ワード)

フォーマット$9のスタック・フレーム

図7.8にフォーマット＄9のスタック・フレームを示す．68020はこれをコプロセッサの事中命令に関する例外で用いる．

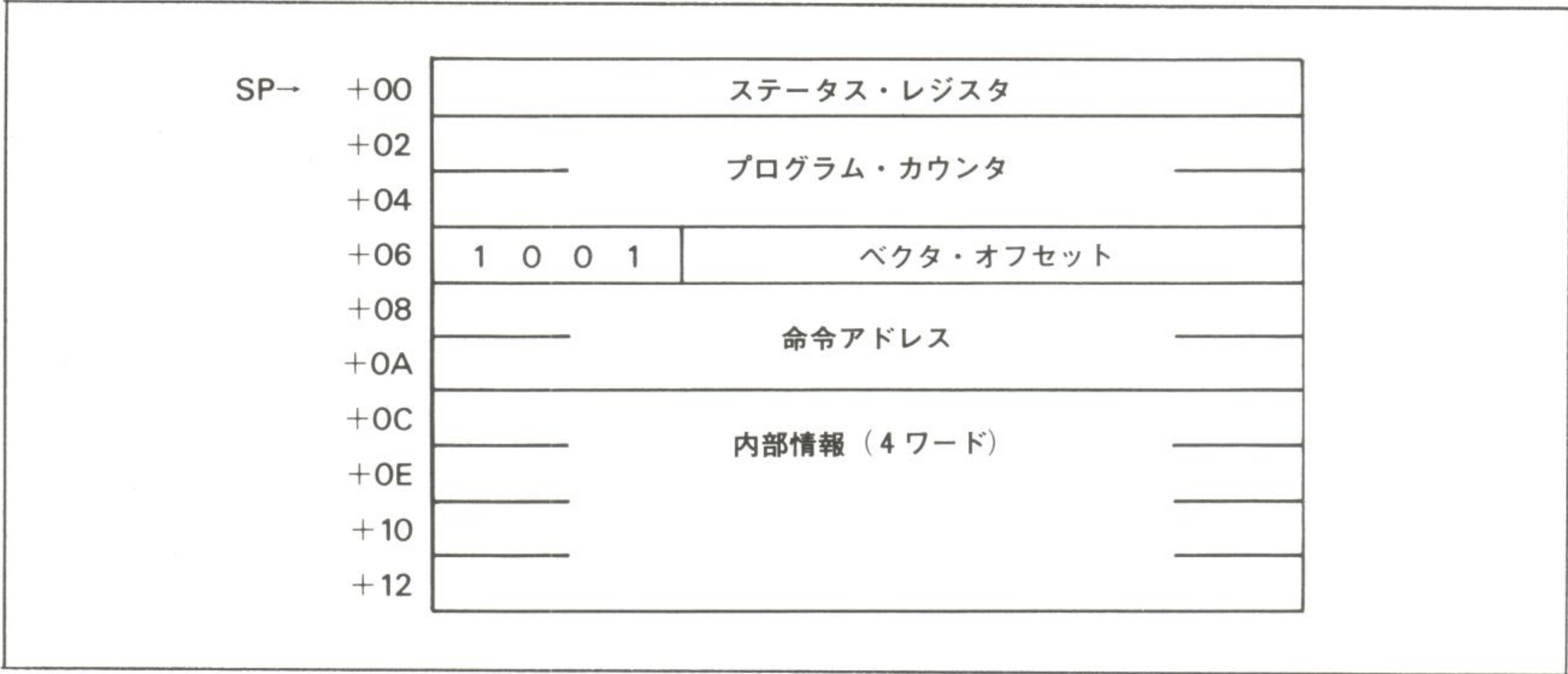



図7.8　フォーマット＄9のスタック・フレーム（10ワード）

フォーマット$Aのスタック・フレーム

図7.9にフォーマット＄Aのスタック・フレームを示す．68020はこれを命令境界におけるバス・サイクル・エラーに関して用いる．

オフセット	内容
SP→ +00	ステータス・レジスタ
+02	プログラム・カウンタ
+04	
+06	1 0 1 0 / ベクタ・オフセット
+08	内部情報
+0A	ステータス・ワード
+0C	命令パイプ・ステージC
+0E	命令パイプ・ステージB
+10	データ・サイクル・フォールト・アドレス
+12	
+14	内部情報
+16	
+18	データ出力バッファ
+1A	
+1C	内部情報
+1E	

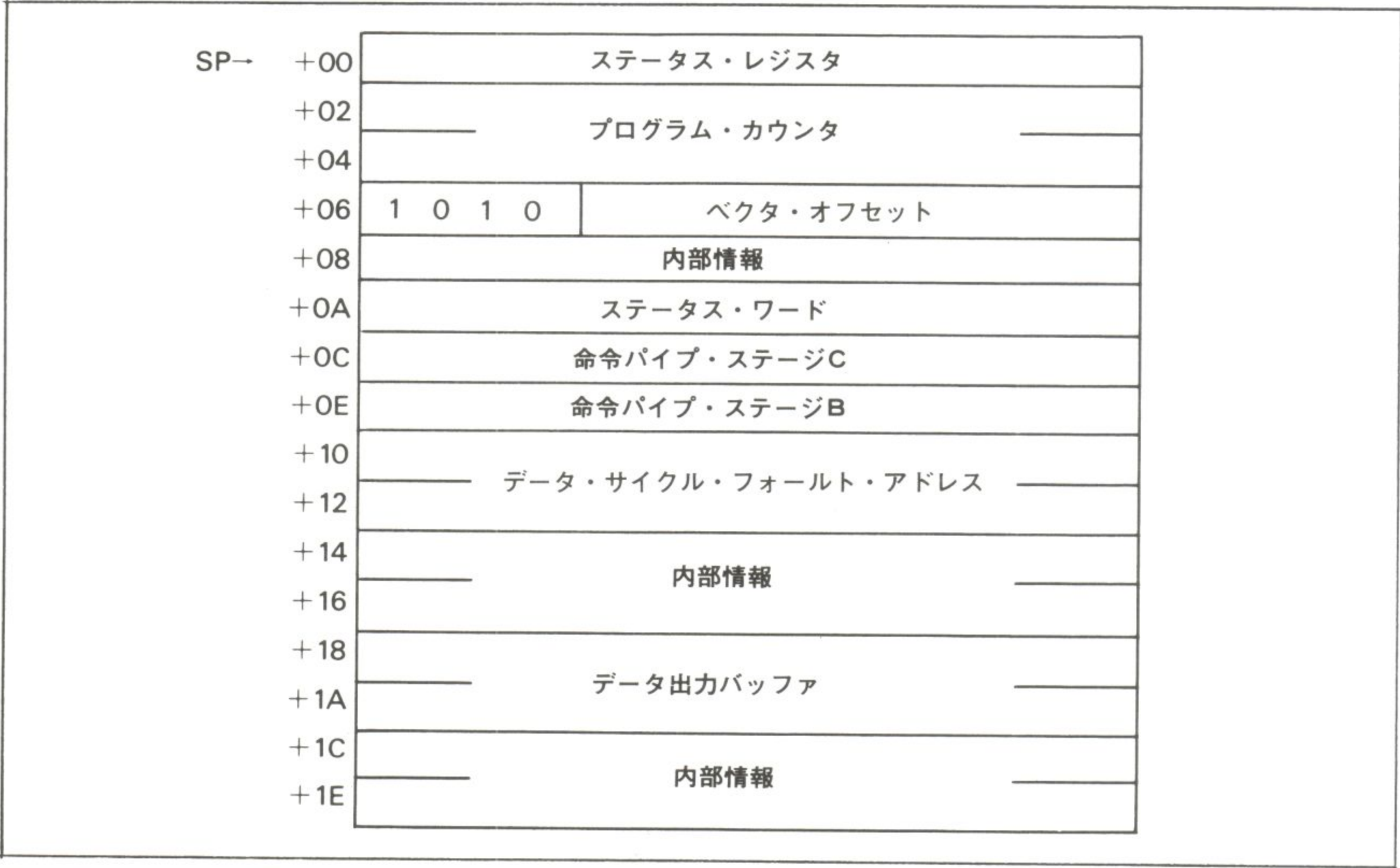



図7.9　フォーマット＄Aのスタック・フレーム（16ワード）

フォーマット$Bのスタック・フレーム

図7.10にフォーマット＄Bのスタック・フレームを示す．68020はこれを命令の実行中におけるバス・サイクル・エラーに関して用いる．前のものよりは命令実行に関係した内部情報が多いが，プロセッサの状態を保存するためにはそれらをセーブしなければならない．

図7.10 フォーマット＄Bのスタック・フレーム(44ワード)

例外処理シーケンス

例外のそれぞれの型に対する一般的な動作シーケンスは同じである．この一般的なシーケンスは次のとおりである．

1．ステータス・レジスタの内容をコピーし，ステータス・レジスタのＳビットのセットとＴビット*のクリアを行う．また割込みの場合は割込みマスクを変更する．
2．ベクタ・テーブルの入口を定める．これは割込みデバイスからの読出しによるか，そうでなければ他の型の例外処理に対応する特定入口番号を用いて定める．
3．適当なデータをスーパバイザ・スタックに押し込む．このデータ型は例外の型と 68K によって定まる．
4．ベクタ・テーブルからアドレスをプログラム・カウンタ PC にロードして実行を始める．

命令トラップ

命令トラップは特定の命令を実行した結果生ずる．この命令には TRAP 命令のように必ず例外処理を起こすものと，TRAPcc, TRAPV, cpTRAPcc, CHK, CHK 2，DIVS，DIVU，CALLM，RTM などの命令のように，それを実行して異常事態が発生したときだけ例外処理を起こすものとがある．

68000 と 68008 では，図7.2に示したように３ワードのスタック・フレームを作る．また 68010 と 68012 では，図7.4に示したように４ワード（フォーマット＄０）のスタック・フレームを作る．68020 のスタック・フレームは命令によって異なり，TRAP 命令ではフォーマット＄０のスタック・フレーム（図7.4参照）を，他の命令ではフォーマット＄２のスタック・フレーム（図 7.6 参照）を用いる．

不当命令と未実装命令

こういう命令に関する例外処理は，命令トラップの場合と同様である．68000 と 68008 では３ワードのスタック・フレーム(図7.2)を用いるが，68010，68012，68020 ではフォーマット＄０のスタック・フレーム（図7.4）を用いる．

＊　68020 ではＴ０，Ｔ１ビット．

不当命令と未実装命令の違いについてまとめよう．AラインやFラインの命令コード（2進法の1010と1111で始まる命令）は未実装命令と考える．このそれぞれに対してベクタ・テーブルの入口があり，未実装命令をエミュレートするのに都合よくできている．

68020では，Fライン命令をコプロセッサ命令として使うように考えている．命令自身の実行不能が確認された後で，アドレス・バス上にコプロセッサの識別情報をエンコードし，プロセッサはCPU空間のバス・サイクル(図5.6)に入る．どのコプロセッサも応答しなければ，外部ロジックがバス・エラーを表示し，プロセッサはFライン・ベクタの入口を利用して例外処理を始める．

ソフトウェアによって未実装命令をエミュレートするのであれば，スタック・フレームに蓄えたプログラム・カウンタPCの内容が不当命令を示していることに留意されたい．エミュレートが終わった後，スタック・フレーム内のPCの内容が修正されたことを確める必要がある．これによって，不当命令に続く次の命令が指示される．

アドレス・エラー

前述のように，命令をワード境界に置く必要がある．もし奇数アドレスから命令をアクセスしようとすると，それは打ち切られて例外処理に入る．68000と68008では7ワードのスタック・フレームを作り（図7.3），68010と68012ではフォーマット＄8のスタック・フレームを作る（図7.7）．68020ではフォーマット＄Aのスタック・フレームを作る（図7.9）．

バス・エラーやアドレス・エラーやリセットに関する例外処理を行っている間に，アドレス・エラーが発生すると，プロセッサはホールト状態になり外部からリセットしなければならない．

トレース

68Kファミリのどれもシングル・ステップでトレースが行える．シングル・ステップのトレースでは，プロセッサは各命令の実行後に例外処理を始める．ステータス・レジスタのトレース・ビット（68000から68012ではTビット，68020ではT1ビット）がセットしてあると，各命令の実行後にトレースが起こる．

再帰的処理を防止するため，この例外処理でプロセッサはまずトレース・ビットをクリアする必要がある．68000と68008では3ワードのスタック・フレームを作り（図7.2），68010と68012ではフォーマット＄0のスタック・フレ

ームを作る（図7.4）．68020 ではフォーマット＄2のスタック・フレームを作る（図7.6）．

68020 には実行フローの変更に関するトレース機能がある*ステータス・レジスタのT0ビットをセットすると，普通は正常に動作しているが，フローが変わったところで割込みが起こる．こういう命令にはBcc，JSR，BSR などがあり，これを実行した後で例外処理が起こる．

不当命令や未実装命令を実行しようとしてもトレースは起こらない．これは命令のエミュレーションにとって重要である．エミュレーション・ルーチンはスタックにのせたステータス・レジスタの内容のトレース・ビットをリターンする前に調べ，セットしてあればトレースもエミュレートする．

ブレークポイント

あるシステムでは，プログラムにある種の「ブレークポイント」を挿入すると便利なことがある．ハードウェア・エミュレータ・システムが十分有効なためには，このようなブレークポイントに着いたことを，プロセッサが外部ハードウェアに教える方法をもたねばならない．68000 と 68008 では不当命令をブレークポイントとして挿入することができた．外部ハードウェアは，ベクタ・テーブルにある不当命令のベクタのアクセスを求めて，アドレス・ラインを監視できる．

68010，68012，68020 にはベクタ・ベース・レジスタ（VBR）があるので，こうはいかない．ベクタ・テーブルの実際のロケーションをプログラムで自由に変更できるからである．そこで，特定の不当命令の集まりを「ブレークポイント」命令として定めることになる．それが＄4848 から＄484F までのパターンをもつ命令であった．

68010 と 68012 では，最初に「ブレークポイント・バス・サイクル」が起こるが，表5.2のように，これはCPU空間のバス・サイクルに含まれる．ブレークポイント・バス・サイクルが起こった後で，不当命令ベクタによるトラップを実行し，正規の不当命令として実行を続ける．

---

*

T1	T0	
0	0	トレースしない
0	1	BRA，JMP などによるフローの変化をトレースする
1	0	すべての命令のトレース
1	1	未定義，保留

68020でもこれに準ずるが，プロセッサはブレークポイント・バス・サイクルを起こすだけでなく，このサイクルの間にCPU空間のブレークポイント番号に相当するアドレスで読取りを行う．これがバス・エラーによって終了すれば，プロセッサは不当命令ベクタによるトラップを実行する．しかし外部ハードウェアは新しい命令をデータ・バスに置き，$\overline{\mathrm{DSACK}x}$信号でサイクルを終わりにすることができる．この場合，プロセッサはデータ・バス上の命令で(パイプライン内の)ブレークポイント命令を置き換え，それで実行を始める．

フォーマット・エラー

68020にはデータのエラー・チェックを行う命令が，CALLM（モジュールを呼ぶ)，RTM（モジュールから戻る)，cpRESTORE（コプロセッサの内部状態を元どおりにする）と3つある．この種の命令は，スタック・フレーム，モジュール・デスクリプタ，フォーマット・フィールドなどで特定データを求めようとし，正しいデータが見つからなかったらフォーマット・エラー例外ベクタによりトラップを起こす．スタック・フレームのフォーマットは＄0である（図7.4)．

コプロセッサ・インターフェースについては**付録B**でもう一度述べる．

割込み

外部デバイスは，割込み要求ライン（$\overline{\mathrm{IPL}\,0}$-$\overline{\mathrm{IPL}\,0}$）を何本かアサートすることによって，プロセッサの正常なフローに割り込める．プロセッサは正常な命令と命令の間でこの要求を認識するが，68020のコプロセッサ命令の中にはその中間で認識するものもある.[*1]割込み要求を受け取ると，$\overline{\mathrm{IPL}\,0}$-$\overline{\mathrm{IPL}\,2}$にある2進数とステータス・レジスタの割込みマスクとを比較し，$\overline{\mathrm{IPL}\,0}$-$\overline{\mathrm{IPL}\,2}$の方が高いかまたは，ノン・マスカブル割込みで$\overline{\mathrm{IPL}\,0}$-$\overline{\mathrm{IPL}\,2}$がすべてアサートされていれば，プロセッサは割込み処理に入る.[*2]そうでなければ，この要求は無視されて正常な処理が続く．

プロセッサが割込みに対して肯定応答しようとすると，アドレス・バスのA1-A3に割込み優先度を出力してCPU空間のバス・サイクルを走らせる(表5.2参照)．デバイスはプロセッサに送るベクタ・テーブルの入口番号を選ぶため，$\overline{\mathrm{DTACK}}$（または$\overline{\mathrm{DSACK}\,0}$/$\overline{\mathrm{DSACK}\,1}$）をアサートし，ベクタ番号を

＊1　事中命令という．
＊2　本来のフローはペンディングになる．

データ・バスにロードする.

またその代わりとして，オートベクタ入口を用いることもある.*1 この場合68000-68012では$\overline{\mathrm{VPA}}$を，68020では$\overline{\mathrm{AVEC}}$をアサートしなければならない.プロセッサは，識別した割込みレベルに応じたオートベクタ入口を用いてトラップする.

割込みアクノリッジ・サイクルに応答するデバイスがない場合は，外部ハードウェアは$\overline{\mathrm{BERR}}$をアサートし，プロセッサはスプリアス割込みベクタ・アドレスを用いてトラップする.

最近の68K周辺デバイスの大部分はプログラム可能な割込みベクタ番号をもっており，割込みアクノリッジでプロセッサにそれを送る．もし番号がない場合は，ソフトウェア的に初期化する場合を除いて，デバイスはベクタ$0Fを送る．このベクタを未初期化ベクタ入口として定めているので，初期の説明書では，「保留」となっていた.*2

割込み例外に対して作られるスタック・フレームもプロセッサで異なっている．68000と68008では3ワードのスタック・フレームを作り（図7.2），68010と68012ではフォーマット＄0のスタック・フレームを作る（図7.4）．68020では割込みに応じてプロセッサが何をするかで定まる．命令トラップの場合ならフォーマット＄0のスタック・フレームを用い（図7.4），コプロセッサ命令ではフォーマット＄9のスタック・フレームを用いた（図7.8）.

68020におけるスタック・フレームの先頭番地は，ステータス・レジスタのMビットに関係しており，クリアされていれば，プロセッサはスタック・フレームを割込みスタックに作る．セットしてあれば，プロセッサはマスタ・スタックにスタック・フレームを作り，割込みスタックには「スローアウェイ・スタック・フレーム」（フォーマット＄1）を作る.

バス・エラー

読取り要求や書込み要求に応じて，外部ロジックはデータ転送を行い，$\overline{\mathrm{DTACK}}$（または$\overline{\mathrm{DSACK\,0}}$/$\overline{\mathrm{DSACK\,1}}$）をアサートするか，さもなければバス・エラー信号（$\overline{\mathrm{BERR}}$）で答える．メモリが実際にはなかったかまたは要求したメモリがユーザのアドレス空間になかった場合は，外部ロジックが$\overline{\mathrm{BERR}}$をアサートする．こういうこともあって，メモリ・スロットを有効に利用するため，メモリ管理ユニットを用いるシステムが多い.

＊1　割り込んだデバイスがベクタ番号を生成しなかった場合を指す.

＊2　このような変化は他にもある．ベクタ番号$0Cから$0Eも68000と68020では違う.

68000 と 68008 では，プロセッサは 7 ワードのスタック・フレームをスタックに作る（図7.3）. なお，スタックにおけるプログラム・カウンタ（の内容）は，バス・エラーを起こしたワードの次のワードのアドレスを含んでいる.

仮想メモリ

68010，68012，68020 では「仮想メモリ」が実現できる. 仮想メモリを実現したシステムでは，物理的に利用できるメモリ空間よりも，タスクのためのアドレス空間全体の方が大きくなれる. ある命令コードを実行している間に，オペレーティング・システムはそれ以外のデータや命令コードを（ディスクのような）補助記憶装置に蓄えるようにできる. アンロードしたコードやデータなどが必要になったら，オペレーティング・システムにより，本質的（に必要）でないデータをディスクに急いでコピーし，必要な部分を読み込めばよい.

仮想メモリを有効に実現するためには，プロセッサは要求メモリのないことを確認し，データをディスクから移し，プログラムを続けて実行することができなければならない. 外部ロジックは要求メモリの存在しないことを確認すると $\overline{\mathrm{BERR}}$ をアサートする. 68000 と 68008 では，データがメモリで利用できるようになったとき命令が再スタートできるように，スタックに十分な情報を積むことが難しい.

68010，68012，68020 では，バス・エラーの例外処理を始めるにあたって，もっと綿密なスタック・フレームが作られる. 68010 と 68012 ではフォーマット＄8 のスタック・フレームが作られる（図7.7）. 68020 では，命令フェッチ中のバス・エラーに対してはフォーマット＄A のスタック・フレーム（図7.9）が，命令実行中のバス・エラーに対してはフォーマット＄B のスタック・フレーム（図7.10）が作られる.

スタックに追加されたデータは，プロセッサの内部情報を含んでいる. 例外ルーチンの一部として，プロセッサはスタックにある情報を評価し，ディスクにあるデータの要求によりエラーが起きたかどうかを定める. もしそうであれば，メモリにあるデータを必要なデータと交換すればよい. 適当にメモリ管理レジスタを更新した後，オペレーティング・システムは「障害のあるプログラム」をもう一度アクセスすることができる.

そうするためには，**RTE**（例外からのリターン）命令を出せばよい. RTE 命令はスタック・フレームのフォーマットを眺め，スタックしたデータがバス・エラーによるものであれば，スタックに積み上げた内部データの復旧とプログラム・カウンタの内容を指定してプログラムを再実行する. この場合，プログラムの継続点は命令の途中でも最初でもよい.

ダブル・バス・フォールト

どの 68K プロセッサでも，ベクタのフェッチやスタック動作中にもう一度バス・フォールトを起こすと，プロセッサはホールト状態になる. これを「ダブル・バス・フォールト」といい，この状態の回復は外部からのリセットしかない.

リセット

いろいろな例外処理の中にあって，一切の情報をスタックする必要のない点で，リセットはユニークである．外部リセットがアサートされると，次のようになる．

1. 実行中の処理が打ち切られる．
2. ステータス・レジスタのSビットがセットされ，68020ではMビットがクリアされる．トレース・ビットもクリアされ，割込みマスクはレベル7にセットされ，68010-68020ではベクタ・ベース・レジスタがクリアされ，68020ではキャッシュ制御レジスタがクリアされる．
3. プロセッサはベクタ・テーブルで＄0000をオフセットとしてロングワードをスーパバイザ・スタック・ポインタ（68020ではISP）にロードし，＄0004をオフセットとしてロングワードでプログラム・カウンタをロードする．このベクタ入口のフェッチはスーパバイザ・プログラム空間で行われる．

   他の例外ベクタはスーパバイザ・データ空間で行われる．
4. プロセッサは$\overline{\text{RESET}}$をアサートして外部デバイスをリセットし，PCを更新して実行を始める．

上述のレジスタ以外のレジスタはこの段階では未定義である．

付　録

付録A

命令の
オブジェクト・コード

(　) 内はアセンブラにおける記法を表す．
ea は実効アドレスの略である．

イミディエイト OR (ORI＃〈データ〉,〈ea〉)

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	0	0	0	0	サイズ		実効アドレス					
										モード			レジスタ		

サイズ：00＝バイト　01＝ワード　10＝ロングワード

CCR とのイミディエイト OR (ORI＃〈データ〉, CCR)

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	0	0	0	0	0	0	1	1	1	1	0	0
0	0	0	0	0	0	0	0	バイト・データ							

SR とのイミディエイト OR (ORI＃〈データ〉, SR)

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	0	0	0	0	0	1	1	1	1	1	0	0
ワード・データ															

CMP 2 (MC 68020) (CMP 2 〈ea〉, Rn)

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	0	サイズ		0	1	1	実効アドレス					
										モード			レジスタ		
A/D	レジスタ			0	0	0	0	0	0	0	0	0	0	0	0

サイズ：00＝バイト　01＝ワード　10＝ロングワード
A/D　0：D (データ)　1：A (アドレス)

CHK 2 （MC 68020）（CHK 2 〈ea〉, Rn）

<table>
<tr><td>15</td><td>14</td><td>13</td><td>12</td><td>11</td><td>10</td><td>9</td><td>8</td><td>7</td><td>6</td><td>5</td><td>4</td><td>3</td><td>2</td><td>1</td><td>0</td></tr>
<tr><td rowspan="2">0</td><td rowspan="2">0</td><td rowspan="2">0</td><td rowspan="2">0</td><td rowspan="2">0</td><td rowspan="2" colspan="2">サイズ</td><td rowspan="2">0</td><td rowspan="2">1</td><td rowspan="2">1</td><td colspan="6">実効アドレス</td></tr>
<tr><td colspan="3">モード</td><td colspan="3">レジスタ</td></tr>
<tr><td>A/D</td><td colspan="3">レジスタ</td><td>1</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td></tr>
</table>

サイズ：00＝バイト　01＝ワード　10＝ロングワード
A/D　0：D（データ）　1：A（アドレス）

ビットの動的変更（BCHG, BCLR, BSET, BTST）（OP　Dn, 〈ea〉）

<table>
<tr><td>15</td><td>14</td><td>13</td><td>12</td><td>11</td><td>10</td><td>9</td><td>8</td><td>7</td><td>6</td><td>5</td><td>4</td><td>3</td><td>2</td><td>1</td><td>0</td></tr>
<tr><td rowspan="2">0</td><td rowspan="2">0</td><td rowspan="2">0</td><td rowspan="2">0</td><td rowspan="2" colspan="3">データ・レジスタ</td><td rowspan="2">1</td><td rowspan="2" colspan="2">タイプ</td><td colspan="6">実効アドレス</td></tr>
<tr><td colspan="3">モード</td><td colspan="3">レジスタ</td></tr>
</table>

タイプ：00＝TST　10＝CLR
　　　　01＝CHG　11＝SET

MOVEP（MOVEP　Dx, d(Ay)；MOVEP　d(Ay), Dx）

<table>
<tr><td>15</td><td>14</td><td>13</td><td>12</td><td>11</td><td>10</td><td>9</td><td>8</td><td>7</td><td>6</td><td>5</td><td>4</td><td>3</td><td>2</td><td>1</td><td>0</td></tr>
<tr><td>0</td><td>0</td><td>0</td><td>0</td><td colspan="3">データ・レジスタ</td><td colspan="3">OP モード</td><td>0</td><td>0</td><td>1</td><td colspan="3">アドレス・レジスタ</td></tr>
</table>

OP モード：100＝メモリからレジスタへのワード転送
　　　　　101＝メモリからレジスタへのロングワード転送
　　　　　110＝レジスタからメモリへのワード転送
　　　　　111＝レジスタからメモリへのロングワード転送

イミィデエイトAND（ANDI＃〈データ〉, 〈ea〉）

<table>
<tr><td>15</td><td>14</td><td>13</td><td>12</td><td>11</td><td>10</td><td>9</td><td>8</td><td>7</td><td>6</td><td>5</td><td>4</td><td>3</td><td>2</td><td>1</td><td>0</td></tr>
<tr><td rowspan="2">0</td><td rowspan="2">0</td><td rowspan="2">0</td><td rowspan="2">0</td><td rowspan="2">0</td><td rowspan="2">0</td><td rowspan="2">1</td><td rowspan="2">0</td><td rowspan="2" colspan="2">サイズ</td><td colspan="6">実効アドレス</td></tr>
<tr><td colspan="3">モード</td><td colspan="3">レジスタ</td></tr>
</table>

サイズ：00＝バイト　01＝ワード　10＝ロングワード

CCRとのイミディエイトAND（ANDI＃〈データ〉, CCR）

<table>
<tr><td>15</td><td>14</td><td>13</td><td>12</td><td>11</td><td>10</td><td>9</td><td>8</td><td>7</td><td>6</td><td>5</td><td>4</td><td>3</td><td>2</td><td>1</td><td>0</td></tr>
<tr><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>1</td><td>0</td><td>0</td><td>0</td><td>1</td><td>1</td><td>1</td><td>1</td><td>0</td><td>0</td></tr>
<tr><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td colspan="8">バイト・データ</td></tr>
</table>

SRとのイミディエイトAND（ANDI＃〈データ〉, SR）

<table>
<tr><td>15</td><td>14</td><td>13</td><td>12</td><td>11</td><td>10</td><td>9</td><td>8</td><td>7</td><td>6</td><td>5</td><td>4</td><td>3</td><td>2</td><td>1</td><td>0</td></tr>
<tr><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>1</td><td>0</td><td>0</td><td>1</td><td>1</td><td>1</td><td>1</td><td>1</td><td>0</td><td>0</td></tr>
<tr><td colspan="16">ワード・データ</td></tr>
</table>

イミディエイトSUB (SUBI#〈データ〉,〈ea〉)

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	0	1	0	0	サイズ		実効アドレス					
										モード			レジスタ		

サイズ：00＝バイト　01＝ワード　10＝ロングワード

イミディエイトADD (ADDI#〈データ〉,〈ea〉)

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	0	1	1	0	サイズ		実効アドレス					
										モード			レジスタ		

サイズ：00＝バイト　01＝ワード　10＝ロングワード

RTM (MC 68020) (RTM　Rn)

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	0	1	1	0	1	1	0	0	A/D	レジスタ		

A/D　0：D（データ）　1：A（アドレス）

CALLM (MC 68020) (CALLM#〈データ〉,〈ea〉)

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	0	1	1	0	1	1	実効アドレス					
										モード			レジスタ		
0	0	0	0	0	0	0	0	ディスプレースメント引数							

A/D　0：D（データ）　1：A（アドレス）

CAS (MC 68020) (CAS　Dc, Du,〈ea〉)

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	1	サイズ		0	1	1	実効アドレス					
										モード			レジスタ		
0	0	0	0	0	0	0	Du			0	0	0	Dc		

サイズ：01＝バイト　10＝ワード　11＝長ワード

CAS 2 (MC 68020) (CAS 2　Dc 1：Dc 2, Du 1：Du 2, (Rn 1)：(Rn 2))

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	1	サイズ		0	1	1	1	1	1	1	0	0
D/A	レジスタRn 1			0	0	0	Du1			0	0	0	Dc1		
D/A	レジスタRn 2			0	0	0	Du2			0	0	0	Dc2		

サイズ：01＝バイト　10＝ワード　11＝ロングワード

ビットの静的変更 (BCHG, BCLR, BSET, STST) (OP#〈データ〉,〈ea〉)

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	1	0	0	0	タイプ		実効アドレス					
										モード			レジスタ		

タイプ：00＝TST　10＝CLR
01＝CHG　11＝SET

イミディエイトEOR (EORI＃〈データ〉, 〈ea〉)

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	1	0	1	0	サイズ		実効アドレス					
										モード			レジスタ		

サイズ：00＝バイト　01＝ワード　10＝ロングワード

CCRとのイミディエイトEOR (EORI＃〈データ〉, CCR)

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	1	0	1	0	0	0	1	1	1	1	0	0
0	0	0	0	0	0	0	0	バイト・データ							

SRとのイミディエイトEOR (EORI＃〈データ〉, SR)

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	1	0	1	0	0	1	1	1	1	1	0	0
ワード・データ															

イミディエイトCMP (CMPI＃〈データ〉, 〈ea〉)

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	1	1	0	0	サイズ		実効アドレス					
										モード			レジスタ		

サイズ：00＝バイト　01＝ワード　10＝ロングワード

MOVES (MOVES Rn, 〈ea〉; MOVES 〈ea〉, Rn)

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	1	1	1	0	サイズ		実効アドレス					
										モード			レジスタ		
A/D	レジスタ			dr	0	0	0	0	0	0	0	0	0	0	0

dr：0＝実効アドレスからレジスタへ
　　1＝レジスタから実効アドレスへ

サイズ：00＝バイト　01＝ワード　10＝ロングワード
A/D　0：D(データ)　1：A(アドレス)

バイトのMOVE (MOVE 〈ea〉, 〈ea〉)

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	1	デスティネーション						ソース					
				レジスタ			モード			モード			レジスタ		

レジスタとモードの位置に注意

ロングワードのMOVEA (MOVEA 〈ea〉, An)

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	1	0	デスティネーション・レジスタ			0	0	1	ソース					
										モード			レジスタ		

ロングワードのMOVE（MOVE 〈ea〉, 〈ea〉）

<table>
<tr><th>15</th><th>14</th><th>13</th><th>12</th><th>11</th><th>10</th><th>9</th><th>8</th><th>7</th><th>6</th><th>5</th><th>4</th><th>3</th><th>2</th><th>1</th><th>0</th></tr>
<tr><td rowspan="2">0</td><td rowspan="2">0</td><td rowspan="2">1</td><td rowspan="2">0</td><td colspan="6">デスティネーション</td><td colspan="6">ソース</td></tr>
<tr><td colspan="3">レジスタ</td><td colspan="3">モード</td><td colspan="3">モード</td><td colspan="3">レジスタ</td></tr>
</table>

レジスタとモードの位置に注意

ワードのMOVEA（MOVEA 〈ea〉, An）

<table>
<tr><th>15</th><th>14</th><th>13</th><th>12</th><th>11</th><th>10</th><th>9</th><th>8</th><th>7</th><th>6</th><th>5</th><th>4</th><th>3</th><th>2</th><th>1</th><th>0</th></tr>
<tr><td rowspan="2">0</td><td rowspan="2">0</td><td rowspan="2">1</td><td rowspan="2">1</td><td colspan="3" rowspan="2">デスティネーション・レジスタ</td><td rowspan="2">0</td><td rowspan="2">0</td><td rowspan="2">1</td><td colspan="6">ソース</td></tr>
<tr><td colspan="3">モード</td><td colspan="3">レジスタ</td></tr>
</table>

ワードのMOVE（MOVE 〈ea〉, 〈ea〉）

<table>
<tr><th>15</th><th>14</th><th>13</th><th>12</th><th>11</th><th>10</th><th>9</th><th>8</th><th>7</th><th>6</th><th>5</th><th>4</th><th>3</th><th>2</th><th>1</th><th>0</th></tr>
<tr><td rowspan="2">0</td><td rowspan="2">0</td><td rowspan="2">1</td><td rowspan="2">1</td><td colspan="6">デスティネーション</td><td colspan="6">ソース</td></tr>
<tr><td colspan="3">レジスタ</td><td colspan="3">モード</td><td colspan="3">モード</td><td colspan="3">レジスタ</td></tr>
</table>

レジスタとモードの位置に注意

NEGX（NEGX 〈ea〉）

<table>
<tr><th>15</th><th>14</th><th>13</th><th>12</th><th>11</th><th>10</th><th>9</th><th>8</th><th>7</th><th>6</th><th>5</th><th>4</th><th>3</th><th>2</th><th>1</th><th>0</th></tr>
<tr><td rowspan="2">0</td><td rowspan="2">1</td><td rowspan="2">0</td><td rowspan="2">0</td><td rowspan="2">0</td><td rowspan="2">0</td><td rowspan="2">0</td><td rowspan="2">0</td><td colspan="2" rowspan="2">サイズ</td><td colspan="6">実効アドレス</td></tr>
<tr><td colspan="3">モード</td><td colspan="3">レジスタ</td></tr>
</table>

サイズ：00＝バイト　01＝ワード　10＝ロングワード

SRからMOVE（MOVE　SR, 〈ea〉）

<table>
<tr><th>15</th><th>14</th><th>13</th><th>12</th><th>11</th><th>10</th><th>9</th><th>8</th><th>7</th><th>6</th><th>5</th><th>4</th><th>3</th><th>2</th><th>1</th><th>0</th></tr>
<tr><td rowspan="2">0</td><td rowspan="2">1</td><td rowspan="2">0</td><td rowspan="2">0</td><td rowspan="2">0</td><td rowspan="2">0</td><td rowspan="2">0</td><td rowspan="2">0</td><td rowspan="2">1</td><td rowspan="2">1</td><td colspan="6">実効アドレス</td></tr>
<tr><td colspan="3">モード</td><td colspan="3">レジスタ</td></tr>
</table>

CHK（CHK 〈ea〉, Dn）

<table>
<tr><th>15</th><th>14</th><th>13</th><th>12</th><th>11</th><th>10</th><th>9</th><th>8</th><th>7</th><th>6</th><th>5</th><th>4</th><th>3</th><th>2</th><th>1</th><th>0</th></tr>
<tr><td rowspan="2">0</td><td rowspan="2">1</td><td rowspan="2">0</td><td rowspan="2">0</td><td colspan="3" rowspan="2">データ・レジスタ</td><td colspan="2" rowspan="2">サイズ</td><td rowspan="2">0</td><td colspan="6">実効アドレス</td></tr>
<tr><td colspan="3">モード</td><td colspan="3">レジスタ</td></tr>
</table>

サイズ：10＝ロングワード（MC68020）
　　　　11＝ワード

LEA（LEA 〈ea〉, An）

<table>
<tr><th>15</th><th>14</th><th>13</th><th>12</th><th>11</th><th>10</th><th>9</th><th>8</th><th>7</th><th>6</th><th>5</th><th>4</th><th>3</th><th>2</th><th>1</th><th>0</th></tr>
<tr><td rowspan="2">0</td><td rowspan="2">1</td><td rowspan="2">0</td><td rowspan="2">0</td><td colspan="3" rowspan="2">アドレス・レジスタ</td><td rowspan="2">1</td><td rowspan="2">1</td><td rowspan="2">1</td><td colspan="6">実効アドレス</td></tr>
<tr><td colspan="3">モード</td><td colspan="3">レジスタ</td></tr>
</table>

CLR (CLR 〈ea〉)

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	0	0	0	1	0	サイズ		実効アドレス					
										モード			レジスタ		

サイズ：00＝バイト　01＝ワード　10＝ロングワード

CCRからMOVE (MC68010) (MOVE　CCR, 〈ea〉)

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	0	0	0	1	0	1	1	実効アドレス					
										モード			レジスタ		

NEG (NEG 〈ea〉)

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	0	0	1	0	0	サイズ		実効アドレス					
										モード			レジスタ		

サイズ：00＝バイト　01＝ワード　10＝ロングワード

CCRへMOVE (MOVE 〈ea〉, CCR)

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	0	0	1	0	0	1	1	実効アドレス					
										モード			レジスタ		

NOT (NOT 〈ea〉)

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	0	0	1	1	0	サイズ		実効アドレス					
										モード			レジスタ		

サイズ：00＝バイト　01＝ワード　10＝ロングワード

SRへMOVE (MOVE 〈ea〉, SR)

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	0	0	1	1	0	1	1	実効アドレス					
										モード			レジスタ		

NBCD (NBCD 〈ea〉)

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	0	1	0	0	0	0	0	実効アドレス					
										モード			レジスタ		

ロングワードのLINK（MC68020）（LINK　An，♯〈ディスプレースメント〉）

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	0	1	0	0	0	0	0	0	0	1	データ・レジスタ		
32ビットディスプレースメント（上位）															
32ビットディスプレースメント（下位）															

SWAP（SWAP　Dn）

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	0	1	0	0	0	0	1	0	0	0	データ・レジスタ		

BKPT（MC68010）（BKPT　♯〈ベクタ〉）

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	0	1	0	0	0	0	1	0	0	1	ベクタ		

PEA（PEA〈ea〉）

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
										実効アドレス					
0	1	0	0	1	0	0	0	0	1	モード			レジスタ		

EXT/EXTB（EXTB-MC68020）（EXT. W　Dn）（EXTB. L　Dn）

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	0	1	0	0	OPモード			0	0	0	データ・レジスタ		

OPモード：010＝バイト──→ワード　011＝ワード──→ロングワード　111＝バイト──→ロングワード（MC68020）

レジスタからEAへのMOVEM（MOVEM　レジスタ・リスト，〈ea〉／MOVEM　〈ea〉，レジスタ・リスト）

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
										実効アドレス					
0	1	0	0	1	0	0	0	1	サイズ	モード			レジスタ		

サイズ：0＝ワード転送　1＝ロングワード転送

TST（TST〈ea〉）

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
										実効アドレス					
0	1	0	0	1	0	1	0	サイズ		モード			レジスタ		

サイズ：00＝バイト　01＝ワード　10＝ロングワード

TAS (TAS 〈ea〉)

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	0	1	0	1	0	1	1	実効アドレス					
										モード			レジスタ		

ILLEGAL (ILLEGAL)

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	0	1	0	1	0	1	1	1	1	1	1	0	0

ロングワードのMULS/MULU (MC 68020)
(MULS/MULU.W 〈ea〉, Dn　16×16→32
MULS/MULU.L 〈ea〉, Dl　32×32→32
MULS/MULU.L 〈ea〉, Dh：Dl　32×32→64)

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	0	1	1	0	0	0	0	実効アドレス					
										モード			レジスタ		
0	Dl			タイプ	サイズ	0	0	0	0	0	0	Dh			

タイプ：0＝MULU　　サイズ：0＝32ビットの積
　　　　1＝MULS　　　　　　1＝64ビットの積

ロングワードのDIVS/DIVU (MC 68020)
DIVUL/DIVSL (MC 68020)
(DIVS.L 〈ea〉, Dr：Dq　64/32→32r：32q
DIVSL.L 〈ea〉, Dr：Dq　32/32→32r：32q
DIVU も同様)

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	0	1	1	0	0	0	1	実効アドレス					
										モード			レジスタ		
0	Dq			タイプ	サイズ	0	0	0	0	0	0	0	Dr		

タイプ：0＝DIVU　　サイズ：0＝ロングワードの被除数
　　　　1＝DIVS　　　　　　1＝クォードワードの被除数

EA からレジスタへの MOVEM
(MOVEM レジスタ・リスト, 〈ea〉
MOVEM 〈ea〉, レジスタ・リスト)

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	0	1	1	0	0	1	サイズ	実効アドレス					
										モード			レジスタ		

サイズ：0＝ワード転送　1＝ロングワード転送

TRAP (TRAP ♯ 〈ベクタ〉)

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	0	1	1	1	0	0	1	0	0	ベクタ			

ワードの LINK（LINK　An，＃〈ディスプレースメント〉）

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	0	1	1	1	0	0	1	0	1	0	アドレス・レジスタ		
ワード・ディスプレースメント															

UNLK（UNLK　An）

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	0	1	1	1	0	0	1	0	1	1	アドレス・レジスタ		

USP へ MOVE（MOVE　An，USP）

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	0	1	1	1	0	0	1	1	0	0	アドレス・レジスタ		

USP から MOVE（MOVE　USP，An）

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	0	1	1	1	0	0	1	1	0	1	アドレス・レジスタ		

RESET（RESET）

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	0	1	1	1	0	0	1	1	1	0	0	0	0

NOP（NOP）

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	0	1	1	1	0	0	1	1	1	0	0	0	1

STOP（STOP＃〈データ〉）

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	0	1	1	1	0	0	1	1	1	0	0	1	0

RTE（RTE）

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	0	1	1	1	0	0	1	1	1	0	0	1	1

RTD（**MC 68010**）（RTD＃〈ディスプレーメント〉）

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	0	1	1	1	0	0	1	1	1	0	1	0	0

RTS（RTS）

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	0	1	1	1	0	0	1	1	1	0	1	0	1

TRAPV（TRAPV）

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	0	1	1	1	0	0	1	1	1	0	1	1	0

RTR（RTR）

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	0	1	1	1	0	0	1	1	1	0	1	1	1

MOVEC（**MC 68010，MC 68020**）（MOVEC　Rc，Rn / MOVEC　Rn，Rc）

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	0	1	1	1	0	0	1	1	1	1	0	1	dr
A/D	レジスタ			制御レジスタ											

dr：0＝制御レジスタからレジスタへ
　　1＝レジスタから制御レジスタへ

制御レジスタ：$000＝SFC　　$801＝VBR
$001＝DFC　　$802＝CAAR
$002＝CACR　　$803＝MSP
$800＝USP　　$804＝ISP

JSR（JSR〈ea〉）

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	0	1	1	1	0	1	0	実効アドレス					
										モード			レジスタ		

JMP（JMP〈ea〉）

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	0	1	1	1	0	1	1	実効アドレス					
										モード			レジスタ		

ADDQ (ADDQ＃〈データ〉,〈ea〉)

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	1	データ			0	サイズ		実効アドレス					
										モード			レジスタ		

データ：3ビット，イミディエイト，1-7はそのまま，0は8を表す。
サイズ：00＝バイト　01＝ワード　10＝ロングワード

Scc (Scc〈ea〉)

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	1	コンディション				1	1	実効アドレス					
										モード			レジスタ		

DBcc (DBcc　Dn,〈ラベル〉)

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	1	コンディション				1	1	0	0	1	データ・レジスタ		

TRAPcc (**MC68020**) (TRAPcc, W＃〈データ〉 / TRAPcc, L＃〈データ〉 / TRAPcc)

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	1	コンディション				1	1	1	1	1	モード		
オペランド															

モード：010＝ワード・オペランド　011＝ロングワード・オペランド　100＝オペランドなし

SUBQ (SUBQ＃〈データ〉,〈ea〉)

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	1	データ			1	サイズ		実効アドレス					
										モード			レジスタ		

データ：3ビット，イミディエイト，1-7はそのまま，0は8を表す。
サイズ：00＝バイト　01＝ワード　10＝ロングワイド

Bcc (Bcc〈ラベル〉)

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	1	0	コンディション				8ビット・ディスプレースメント							
8ビット・ディスプレースメントが$00なら16ビット・ディスプレースメント															
8ビット・ディスプレースメントが$FFなら32ビット・ディスプレースメント															

BRA （BRA 〈ラベル〉）

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	1	0	0	0	0	0	8ビット・ディスプレースメント							
8ビット・ディスプレースメントが$00なら16ビット・ディスプレースメント															
8ビット・ディスプレースメントが$FFなら32ビット・ディスプレースメント															

BSR （BSR 〈ラベル〉）

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	1	0	0	0	0	1	8ビット・ディスプレースメント							
8ビット・ディスプレースメントが$00なら16ビット・ディスプレースメント															
8ビット・ディスプレースメントが$FFなら32ビット・ディスプレースメント															

MOVEQ （MOVEQ ＃〈データ〉, Dn）

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	1	1	データ・レジスタ			0	データ							

データ：データを32ビットに符号拡張し，
　　　　データレジスタへ転送する。

OR （OR 〈ea〉, Dn ／ OR Dn, 〈ea〉）

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	0	0	データ・レジスタ			OPモード			実効アドレス					
										モード			レジスタ		

OPモード：

バイト	ワード	ロングワード	動作
000	001	010	(<ea>)v(<Dn>)→<Dn>
100	101	110	(<Dn>)v(<ea>)→<ea>

ワードのDIVU/DIVS （DIVU.W 〈ea〉, Dn　32/16→16r：16q ／ DIVU.L 〈ea〉, Dq　32/32→32q ／ DIVSも同様）

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	0	0	データ・レジスタ			タイプ	1	1	実効アドレス					
										モード			レジスタ		

タイプ：0＝DIVU　1＝DIVS

SBCD （SBCD Dx, Dy ／ SBCD －(Ax), －(Ay)）

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	0	0	デスティネーション・レジスタ*			1	0	0	0	0	R/M	レジスタ* Dx/Ax		

R/M：0＝データ・レジスタからデータ・レジスタへ
　　　1＝メモリからメモリへ
*R/M＝0ならデータ・レジスタ
　R/M＝1ならプリデクリメント・アドレッシング・モードのアドレス・レジスタ

PACK (MC68020) (PACK　−(Ax), −(Ay), ＃〈整合子〉
PACK　Dx, Dy, ＃〈整合子〉)

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	0	0	デスティネーション・レジスタ*			1	0	1	0	0	R/M	ソース・レジスタ*		
16ビットの整合子															

R/M： 0＝データ・レジスタからデータ・レジスタへ
　　　1＝メモリからメモリへ
＊R/M＝0ならデータ・レジスタ
　R/M＝1ならプリデクリメント・アドレッシング・モードのアドレス・レジスタ

UNPK (MC68020) (UNPACK　−(Ax), −(Ay), ＃〈整合子〉
UNPAK　Dx, Dy, ＃〈整合子〉)

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	0	0	デスティネーション・レジスタ*			1	1	0	0	0	R/M	ソース・レジスタ*		

R/M： 0＝データ・レジスタからデータ・レジスタへ
　　　1＝メモリからメモリへ
＊R/M＝0ならデータ・レジスタ
　R/M＝1ならプリデクリメント・アドレッシング・モードのアドレス・レジスタ

SUB (SUB　〈ea〉, Dn
SUB　Dn, 〈ea〉)

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	0	1	データ・レジスタ			OPモード			実効アドレス					
										モード			レジスタ		

OPモード：	バイト	ワード	ロングワード	動作
	000	001	010	(<ea>)−(<Dn>)→<Dn>
	100	101	110	(<Dn>)−(<ea>)→<ea>

SUBA (SUBA 〈ea〉, An)

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	0	1	データ・レジスタ			OPモード			実効アドレス					
										モード			レジスタ		

OPモード：	ワード	ロングワード	動作
	011	111	(<ea>)−(<An>)→<An>

SUBX (SUBX　Dx, Dy
SUBX　−(Ax), −(Ay))

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	0	1	デスティネーション・レジスタ*			1	サイズ		0	0	R/M	ソース・レジスタ*		

サイズ：00＝バイト　01＝ワード　10＝ロングワード
R/M： 0＝データ・レジスタからデータ・レジスタへ　1＝メモリからメモリへ
＊R/M＝0ならデータ・レジスタ
　R/M＝1ならプリデクリメント・ドレッシング・モードのアドレス・レジスタ

CMP (CMP 〈ea〉, Dn)

CMPA (CMPA 〈ea〉, An)

EOR (EOR Dn, 〈ea〉)

CMPM (CMPM (Ay)+, (Ax)+)

サイズ：00＝バイト　01＝ワード　10＝ロングワード

AND (AND 〈ea〉, Dn / AND Dn, 〈ea〉)

ワードのMULU
ワードのMULS (MULS.W 〈ea〉, Dn 16×16→32 / MULU も同様)

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	0	0	データ・レジスタ			タイプ	1	1	実効アドレス モード			実効アドレス レジスタ		

タイプ：0＝MULU　1＝MULS

ABCD (ABCD　Dy, Dx / ABCD　−(Ay), −(Ax))

15	14	13	12	11 10 9	8	7	6	5	4	3	2 1 0
1	1	0	0	デスティネーション・レジスタ*Rx	1	0	0	0	0	R/M	ソース・レジスタ*Ry

R/M：0＝データ・レジスタからデータ・レジスタへ　1＝メモリからメモリへ
＊R/M＝0ならデータ・レジスタ
　R/M＝1ならプリデクリメント・アドレッシング・モードのアドレスレジスタ

データ・レジスタのEXG (EXG　Dx, Dy)

15	14	13	12	11 10 9	8	7	6	5	4	3	2 1 0
1	1	0	0	データ・レジスタDx	1	0	1	0	0	0	データ・レジスタDy

アドレス・レジスタのEXG (EXG　Ax, Ay)

15	14	13	12	11 10 9	8	7	6	5	4	3	2 1 0
1	1	0	0	アドレス・レジスタAx	1	0	1	0	0	1	アドレス・レジスタAy

データ・レジスタとアドレス・レジスタのEXG (EXG　Dx, Ay)

15	14	13	12	11 10 9	8	7	6	5	4	3	2 1 0
1	1	0	0	データ・レジスタDx	1	1	0	0	0	1	アドレス・レジスタAy

ADD (ADD　〈ea〉, Dn / ADD　Dn, 〈ea〉)

15	14	13	12	11 10 9	8 7 6	5 4 3	2 1 0
						実効アドレス	
1	1	0	1	データ・レジスタDn	OPモード	モード	レジスタ

OPモード：	バイト	ワード	ロングワード	動作
	000	001	010	(<ea>)+(<Dn>)→<Dn>
	100	101	110	(<Dn>)+(<ea>)→<ea>

ADDA (ADDA　〈ea〉, An)

15	14	13	12	11 10 9	8 7 6	5 4 3	2 1 0
						実効アドレス	
1	1	0	1	アドレス・レジスタAn	OPモード	モード	レジスタ

OPモード：	ワード	ロングワード	動作
	011	111	(<ea>)+(<An>)→<An>

ADDX (ADDX Dy, Dx / ADDX −(Ay), −(Ax))

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	0	1	ディスティネーション・レジスタ*Rx			1	サイズ		0	0	R/M	ソース・レジスタ*Ry		

サイズ：00＝バイト　01＝ワード　10＝ロングワード
R/M：0＝データ・レジスタからデータ・レジスタへ　1＝メモリからメモリへ
*R/M＝0 ならデータ・レジスタ
R/M＝1 ならプリデクリメント・アドレッシング・モードのアドレス・レジスタ

シフトとローテイト－レジスタ (ASL Dx, Dy / ASL# 〈カウント〉, Dy など)

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	1	0	カウント/レジスタDx			dr	サイズ		i/r	タイプ		データ・レジスタDy		

カウント/レジスタ：i/r＝0 ならシフト数
i/r＝1 ならシフト数を含んだデータ・レジスタ
dr：0＝右　1＝左
サイズ：00＝バイト　01＝ワード　10＝ロングワード
i/r：0＝イミディエイト　1＝レジスタ
タイプ：00＝算術シフト　10＝拡張 X つきローテイト
01＝論理シフト　11＝ローテイト

シフトとローテイト－メモリ (ASL 〈ea〉 など)

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
										実効アドレス					
1	1	1	0	0	タイプ		dr	1	1	モード			レジスタ		

タイプ：00＝算術シフト　01＝論理シフト　10＝拡張X付きローテイト　11＝ローテイト
dr：0＝右　1＝左

ビット・フィールド (MC68020) (BFCHG 〈ea〉 {オフセット：ビット幅} など8種)

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
										実効アドレス					
1	1	1	0	1	タイプ			1	1	モード			レジスタ		
0	レジスタ			Do	オフセット					Dw	ビット幅				

タイプ：000＝BFTST　100＝BFCLR
001＝BFEXTU　101＝BFFFO
010＝BFCHG　110＝BFSET
011＝BFEXTS　111＝BFINS
レジスタが000なら BFTST, BFCHG, BFCLR, BFSET
Do：0＝オフセットはイミディエイト　1＝オフセットはデータ・レジスタ
Dw：0＝ビット幅はイミディエイト　1＝ビット幅はデータ・レジスタ

コプロセッサの命令

cpGEN (MC68020) cpGEN〈コプロセッサで決まるパラメータ〉

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	1	1	コプロセッサ番号			0	0	0	実効アドレス					
										モード			レジスタ		
コプロセッサによる命令語															

cpScc (MC68020) (cpScc〈ea〉)

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	1	1	コプロセッサ番号			0	0	1	実効アドレス					
										モード			レジスタ		
0	0	0	0	0	0	0	0	0	0	コプロセッサの状態					

cpDBcc (MC68020) (cpDBcc Dn,〈ラベル〉)

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	1	1	コプロセッサ番号			0	0	1	0	0	1	レジスタ		
0	0	0	0	0	0	0	0	0	0	コプロセッサの状態					
ディスプレースメント															

cpTRAPcc (MC68020) (cpTRAPcc / cpTRAPcc #,〈データ〉)

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	1	1	コプロセッサ番号			0	0	1	1	1	1	モード		
0	0	0	0	0	0	0	0	0	0	コプロセッサの状態					
オペランド															

モード：010＝ワード　011＝ロングワード　100＝ディスプレースメント

cpBcc (MC68020) (cpBcc〈ラベル〉)

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	1	1	コプロセッサ番号			0	1	サイズ	コプロセッサの状態					
ディスプレースメント															

サイズ：0＝16ビットディスプレースメント　1＝32ビットディスプレースメント

cpSAVE (MC68020) (cpSAVE〈ea〉)

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	1	1	コプロセッサ番号			1	0	0	実効アドレス					
										モード			レジスタ		

cpRESTORE (MC68020) (cpRESTORE 〈ea〉)

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	1	1	コプロセッサ番号			1	0	1	実効アドレス					
										モード			レジスタ		

コプロセッサ・プリミティブ (MC68020)

CA：再来ビット (come-again bit)
PC：プログラム・カウンタ
dr：方向ビット
SP：PC スキャン・ビット (scanPC)
IA：割込み許可ビット (interrupts allowed bit)
PF：プロセス終了の有無ビット (process finished bit)
TF：条件の真偽ビット (true/false bit)

動作中

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	PC	1	0	0	1	0	0	0	0	0	0	0	0	0	0

コプロセッサの複数レジスタの転送

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
CA	PC	dr	0	0	0	0	1	バイト数							

ステータス・レジスタ転送と PC スキャン

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
CA	PC	dr	0	0	0	1	SP	0	0	0	0	0	0	0	0

スーパバイザ・チェック

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
CA	PC	0	0	0	1	0	0	0	0	0	0	0	0	0	0

コプロセッサのレジスタとメイン・メモリとの転送に関する例外

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
CA	PC	dr	0	0	1	0	1	バイト数							

メイン・プロセッサの複数レジスタとの転送

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
CA	PC	dr	0	0	1	1	0	0	0	0	0	0	0	0	0

動作ワードの転送

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
CA	PC	0	0	0	1	1	1	0	0	0	0	0	0	0	0

オペランドなし

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
CA	PC	0	0	1	0	0	IA	0	0	0	0	0	0	PF	TF

実効アドレスの確定と転送に関する例外

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
CA	PC	0	0	1	0	1	0	0	0	0	0	0	0	0	0

メイン・プロセッサ・レジスタの転送

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
CA	PC	dr	0	1	1	0	0	0	0	0	0	D/A	レジスタ		

メイン・プロセッサ制御レジスタの転送

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
CA	PC	dr	0	1	1	0	1	0	0	0	0	0	0	0	0

システム・スタックの転送

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
CA	PC	0	1	1	1	0	バイト長								

長さ　1，2，4，プロトコル違反

命令ストリーム転送

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
CA	PC	0	0	1	1	1	1	バイト長							

実効アドレスの確定とデータの転送

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
CA	PC	dr	1	0	妥当な実効アドレス			バイト長							

事前命令に関する強制例外

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	PC	0	1	1	1	0	0	ベクタ・オフセット							

事中命令に関する強制例外

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	PC	0	1	1	1	0	1	ベクタ・オフセット							

事後命令に関する強制例外

15	14	13	12	11	10	9	8	7–0
0	PC	0	1	1	1	1	0	ベクタ・オフセット

確定済み実効アドレスへの書込み

15	14	13	12	11	10	9	8	7–0
CA	PC	1	0	0	0	0	0	バイト長

付録B

68K プロセッサの
インターフェース

この付録Bでは，68K プロセッサと6800 ファミリの周辺チップとのインターフェースおよびコプロセッサとのインターフェースについて述べる.

6800ファミリとのインターフェース

Motorola 社が68000 を発表した時点では,16 ビット・データ・バスト**非同期バス制御**という2点で,「うまく使える」ような周辺チップが少なかった.[*] しかし，同社にはその前に開発した一連のプロセッサとして6800 ファミリがあり，そこで利用できる多数の周辺チップがあった．しかも，68000 を基本とするシステムの開発を始めるにあたり，68000 と完全にコンパチブルな周辺チップを待っている余裕がなかったので，68000 や68010，68012 に信号を3つ加え，6800 ファミリのデバイスとインターフェースできるようにした.

すでに述べたように，68K プロセッサでは非同期**ハンドシェイキング**を用いて周辺装置とのデータ転送を行っているが，6800 ファミリではデータの同期転送が必要であった．そこで，68000 に送られる6800 ファミリの信号を用いて，同期ハンドシェイキングをエミュレートできるようにした.

必要な信号は有効メモリ・アドレス信号($\overline{\mathrm{VMA}}$)，有効周辺アドレス信号($\overline{\mathrm{VPA}}$)，イネーブル信号（E）である．同期方式のリード/ライト・サイクルのタイミングを**図B. 1** に示す．68K がアドレス・バスにアドレスを出力してアドレス・ス

＊ 多くは8ビット・データ・バスをもち，同期バス制御を使っていた.

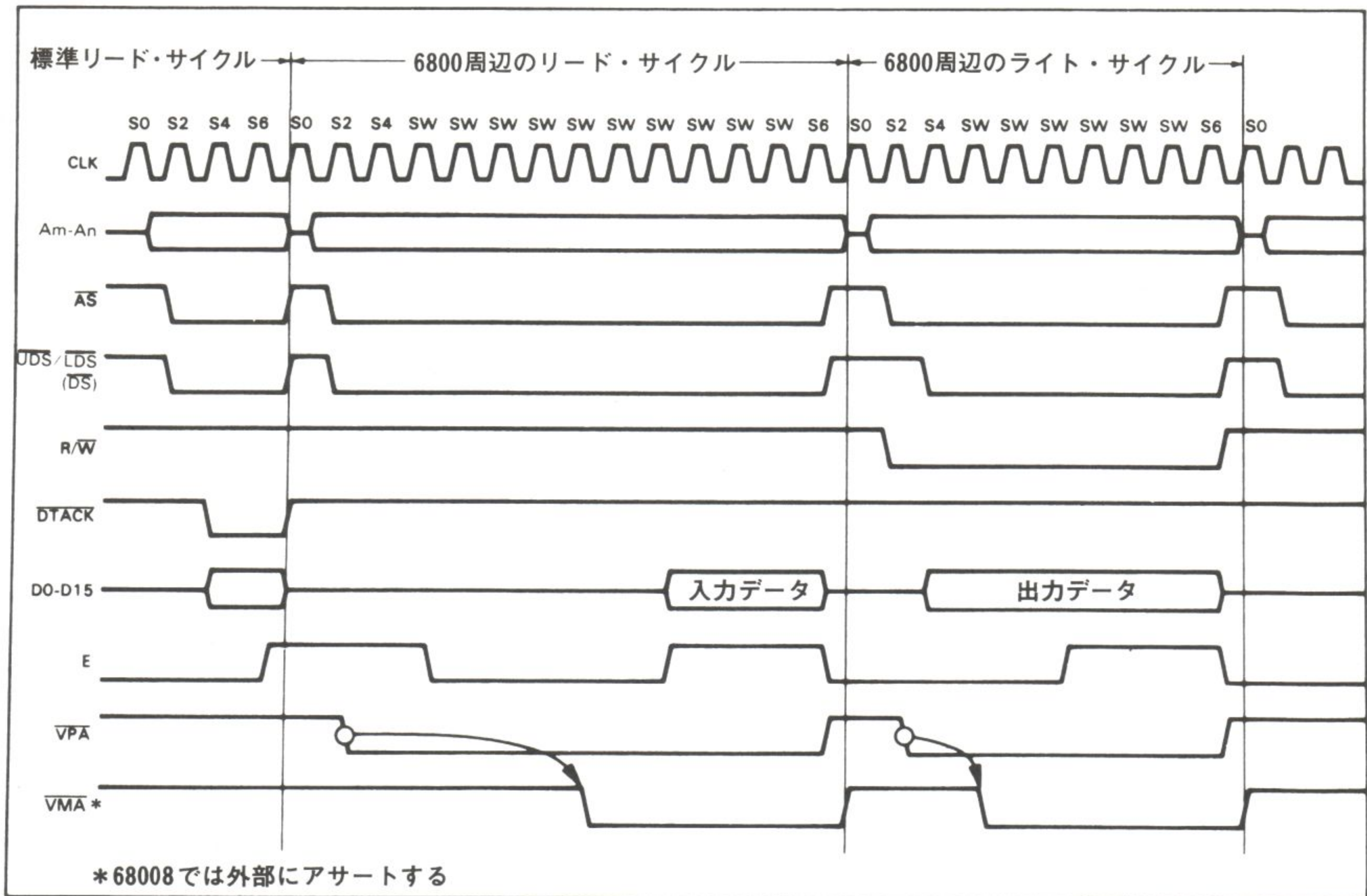



図 B.1 68000の周辺デバイスとの同期リード/ライトのタイミング

トローブをアサートすると，外部ロジックではアドレス・ライン上の情報がデコードされるのを待っている．もし6800の周辺デバイスがアクセスされると外部ロジックは $\overline{\mathrm{VPA}}$ 入力をアサートするが，これは，同期タイミング信号を用いたデータ転送の続行を68Kに知らせている（クロックはE信号による）．

リード・サイクルにおいて，6800の周辺デバイスは，Eがハイのときにデータ・バス上にデータを置くようになっている．Eの立下がりエッジは，非同期転送の場合における $\overline{\mathrm{DTACK}}$ よりも確実に，データ転送の完了を示している．こうして68Kのストローブ信号がネゲートされ，アドレス・バスが高インピーダンス状態に戻り，普通の様式でサイクルが完了したことになる．

6800のリード/ライト・タイミング

図B.1に示したように，リード動作とライト動作とではCLKサイクルの総数が異なる．しかしこれだけで，6800タイプのリード動作がライト動作よりも4CLKサイクルだけ多いと決めてはならない．いつもこうなっている訳ではなく，実際のCLKサイクル数はEの位相に関係している．Eのデューティ・サイクルは40%あり，ハイが4CLKピリオド，ローが6CLKピリオドある．ここに示したライト・サイクルでは，Eは命令実行サイクルと同期しており，この場合のライト・サイクルでは，実行に必要な最小限のCLKサイクルしか取らない．68Kは，Eと同期するために，**ウェイト状態**を自動的に挿入する．

68Kが $\overline{\mathrm{VPA}}$ 信号を受け取ると，周辺装置に $\overline{\mathrm{VMA}}$ 信号を出力する．応用面では，この信号は周辺装置におけるチップ選択入力といろいろな点で関係して

いる．68008では$\overline{\mathrm{VMA}}$信号がないが，外部ロジックは$\overline{\mathrm{VMA}}$をアサートしなければならない．

リード・サイクルやライト・サイクルの最後に，68Kが$\overline{\mathrm{AS}}$をネゲートし，その後1CLKピリオドの範囲で，システムの周辺デバイスやアドレスをデコードするロジックは，$\overline{\mathrm{VPA}}$信号をネゲートしなければならない．そうでないと，次のサイクルも6800タイプのものとみなされる．

応用可能な範囲

最近よく使われる周辺デバイスは，その大部分が非同期転送形式で使えるので，新しいシステムでは6800ファミリのインターフェース信号を使う必要がなくなった．68020のパッケージはこの種の信号を含んでいない．

コプロセッサのインターフェース

プロセッサ68020にはコプロセッサのインターフェースがある．コプロセッサは，主プロセッサの拡張として働く目的をもった特殊なプロセッサである．現在のところ，68020では2つのコプロセッサが使える．* 1つは浮動小数点プロセッサ68881で，他はページ・メモリ管理ユニット68851である．

コプロセッサと普通の周辺プロセッサとでは，プロセッサに対するインターフェースの点で異なる．周辺プロセッサは，MOVEのような普通のプログラム命令で主プロセッサと情報を交換するが，コプロセッサは主プロセッサの一部のように考えてよく，プログラマはアセンブリ言語を用いて情報をわかりやすい形で交換することができる．

68020のコプロセッサ命令は，次図に示すように，Fラインを使う．主プロセッサがFライン命令にであうと，CPU空間サイクルになる．このサイクルの一部として，命令コードに含まれた主プロセッサは，コプロセッサ識別番号(cp-id)をエンコードする．もしこのコプロセッサがないと．外部ロジックは$\overline{\mathrm{BERR}}$をアサートし，主プロセッサはFライン・ベクタを立ててトラップする．

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	1	1	cpid			0	0	0	モード			レジスタ		
コプロセッサ命令															
延長ワード（オプション）															

＊　8個まで接続可能である．

しかし，このコプロセッサがあると，コプロセッサと主プロセッサの間で一連の双方向転送が始まる.* 主プロセッサは，コプロセッサにどのコマンドが命令のOPコードによって示されるように必要となったかを示し，コプロセッサは主プロセッサに必要なオペランドのフェッチを求めて，コマンドを実行する．その後，コプロセッサから主プロセッサに必要なデータを戻すが，コプロセッサでコマンドを実行している間は，主プロセッサはこのコプロセッサをポールし続ける．完了したことがわかると，主プロセッサは本来の命令実行に戻る．

こういう「大きな組織（スキーム）」では，コプロセッサ以外の周辺装置をインターフェースするのと少し方法が異なる．大きな違いは必要なマクロコードおよびマイクロコードにある．すなわち，プロセッサは周辺装置と通信するのに一連の標準命令を用いるが，コプロセッサ・インターフェースに対する実行命令は，完全なマイクロコードである．この実行命令は，外部命令のフェッチをまったく必要としないので，かなり速く実行できる．プログラマもインターフェースを詳しく知る必要がなく，コプロセッサ命令のシンタックスだけを知ればよい．

68K プロセッサの中で，コプロセッサ・インターフェースをマイクロコードで行っているのは 68020 だけである．しかし，インターフェースには制御ラインを，さらに追加しなくてもよいので，他の 68K ファミリは，68020 で示された CPU 空間の転送をエミュレートして，コプロセッサとインターフェースすると考えてよい．このインターフェースの詳細は，本書の範囲外である．68020 で利用できるコプロセッサの動作はユーザにわかりやすいが，特定のコプロセッサ・プリミティブの詳細はここでは扱わない．

---

* もしコプロセッサが稼動中ならば，主プロセッサは待機している．データの転送が必要なら，68020 から転送した後に問い合わせを行う．

付録C

68K ファミリにおける相違点

ここに68K ファミリに含まれる各CPU の相違点をまとめる.

●データ・バスの大きさ（ビット数）

68000	16
68008	8
68010	16
68012	16
68020	8，16，32

●アドレス・バスの大きさ（ビット数）

68000	24
68008	20
68010	24
68012	30（＋A31）
68020	32

●キャッシュ/ループ

68000	なし
68008	なし
68010	3ワード・ループ
68012	3ワード・ループ
68020	128 ワード・キャッシュ

●仮想メモリ／仮想システム

68000	不可

68008	不可
68010	可能
68012	可能
68020	可能

●メモリ・アラインメント

68000	ワード，ロングワード，命令，スタックで偶数アラインメントを必要とする.
68008	ワード，ロングワード，命令，スタックで偶数アラインメントを必要とする.
68010	ワード，ロングワード，命令，スタックで偶数アラインメントを必要とする.
68012	ワード，ロングワード，命令，スタックで偶数アラインメントを必要とする.
68020	命令のみ偶数アラインメントを必要とする.

●制御レジスタ

68000	なし
68008	なし
68010	SFC，DFC，VBR
68012	SFC，DFC，VBR
68020	SFC，DFC，VBR，CACR，CAAR

●スタック・ポインタ

68000	USP，SSP
68008	USP，SSP
68010	USP，SSP
68012	USP，SSP
68020	USP，SSP（MSP，ISP）

●**68020に追加されたアドレッシング・モード**

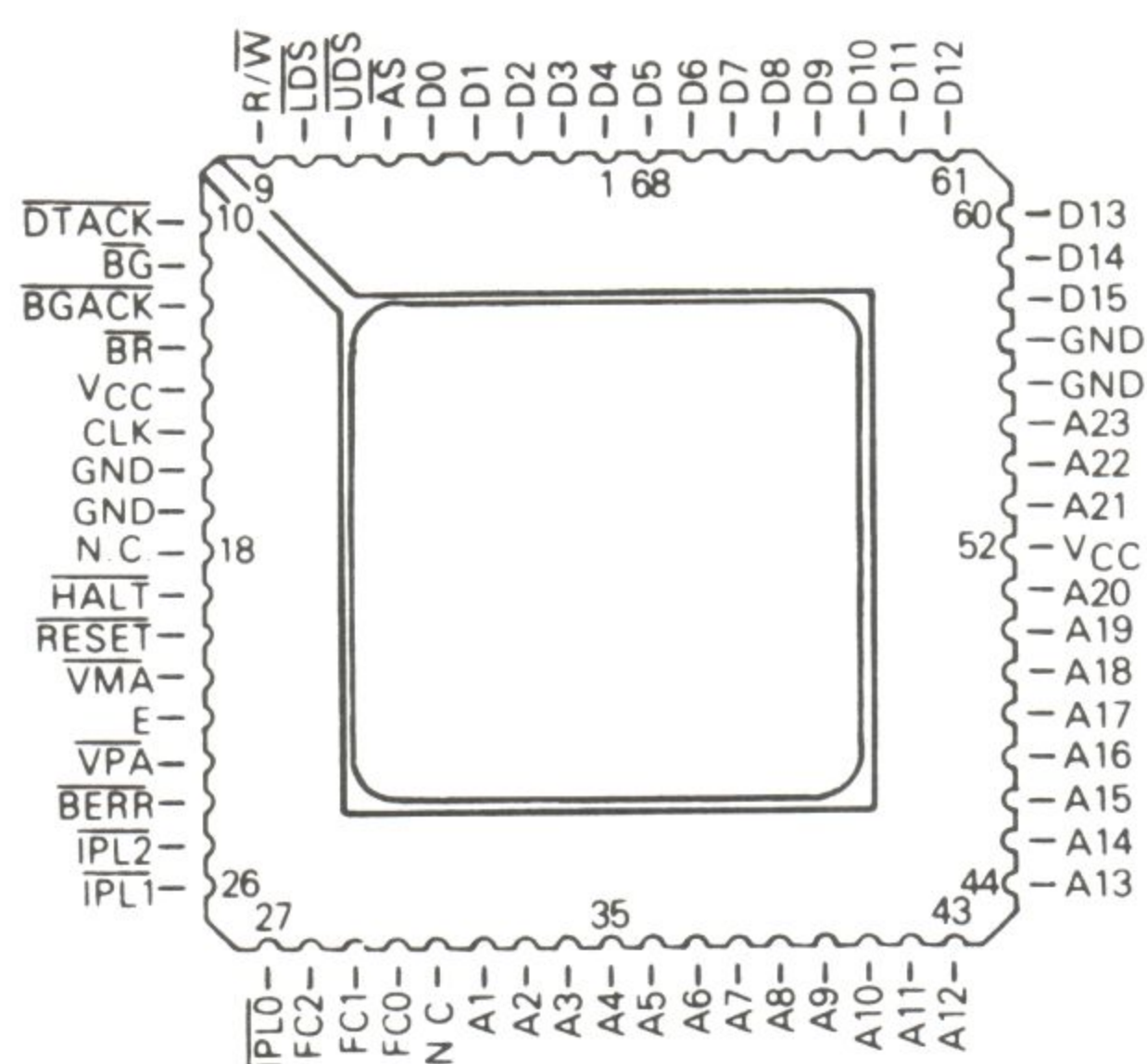
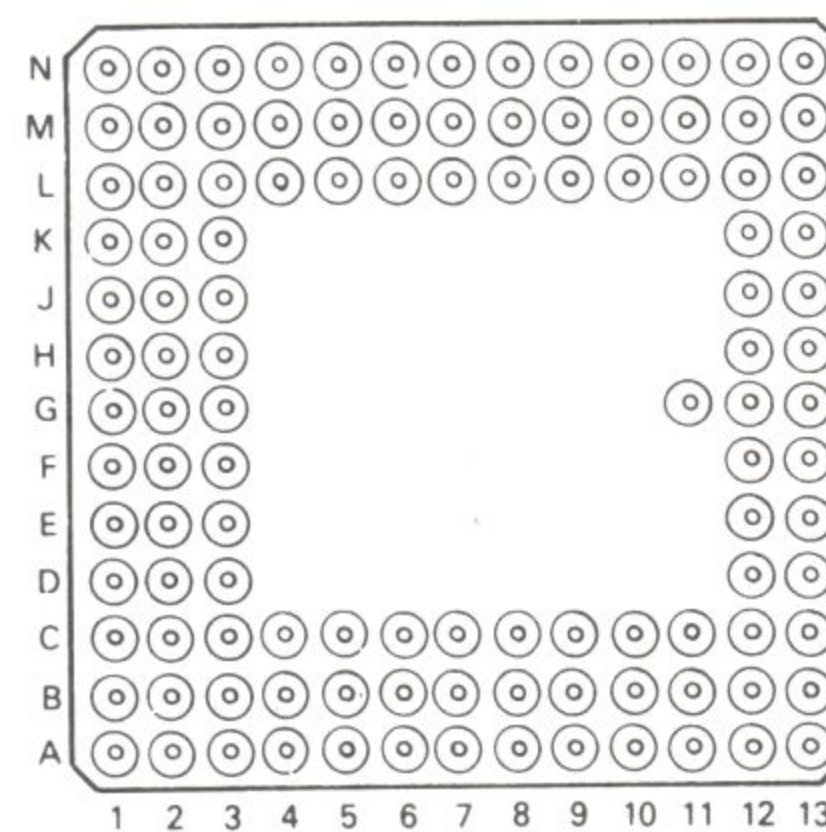
メモリ間接，スケールつき間接，ディスプレースメントの大きさ

●**68020に追加された命令**

Bcc	32ビット・ディスプレースメントをサポートする
BFxx	新しい命令
BKPT	外部操作による置き換えをサポートする
BRA	32ビット・ディスプレースメントをサポートする
BSR	32ビット・ディスプレースメントをサポートする
CALLM	新しい命令
CAS，CAS 2	新しい命令
CHK	32ビット命令をサポートする
CHK 2	新しい命令
CMP 1	新しいアドレッシング・モード
CMP 2	新しい命令
cpxx	新しい命令タイプ
DIVS，DIVU	32ビットと64ビットをオペランドできる
EXTB	8ビット延長を32ビットにできる
LINK	32ビット・ディスプレースメントをサポートする
MOVEC	新しい制御レジスタをサポートする
MULS/MULU	32ビット・オペランドをサポートする
PACK	新しい命令
RTM	新しい命令
TST	新しいアドレッシング・モード
TRAPcc	新しい命令
UNPK	新しい命令

付録D

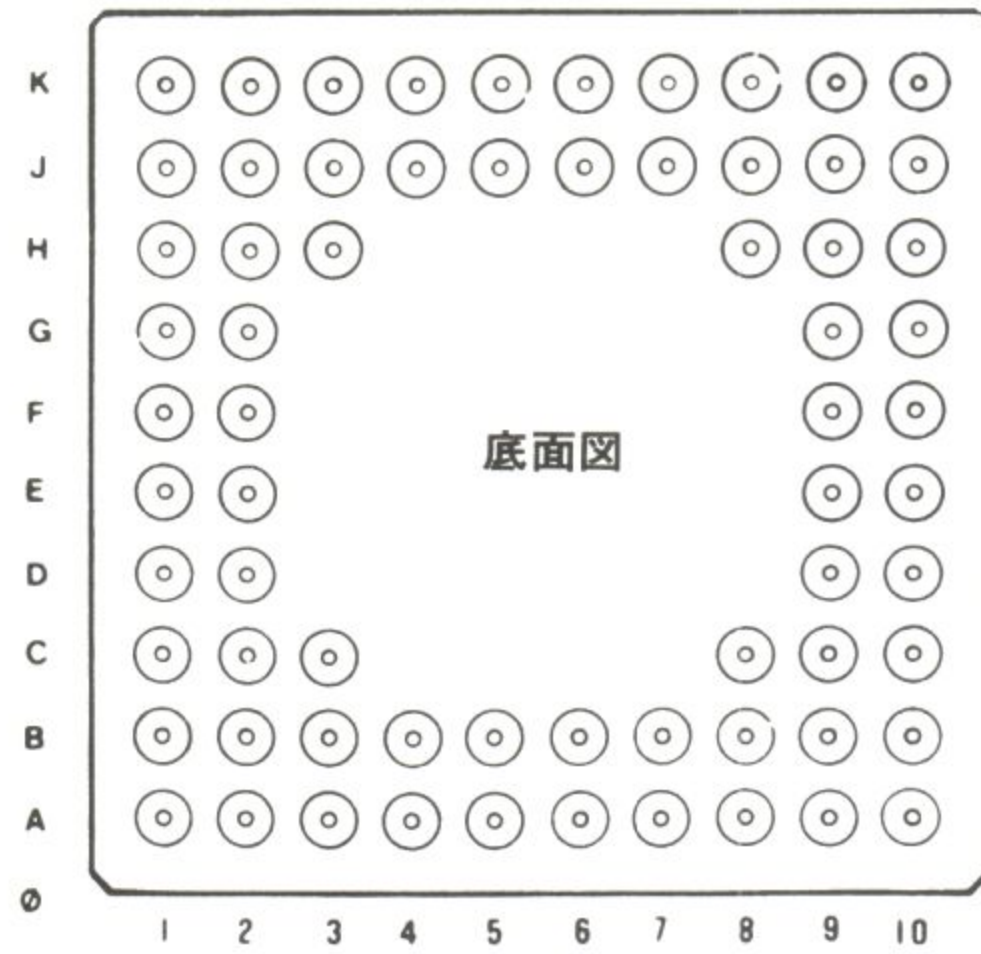
パッケージ

68ピンのチップ・キャリヤ（68000，68010）

114ピン・グリッド・アレイ（68020）

Motorola 社の好意による

68ピン・グリッド・アレイ (68000, 68010)

K J H G F E D C B A ∅

1 2 3 4 5 6 7 8 9 10

底面図

68ピン・グリッド・アレイのピン配置図 (68000, 68010)

Pin Number	Function
A1	接続しない
A2	$\overline{\text{AS}}$
A3	D1
A4	D2
A5	D4
A6	D5
A7	D7
A8	D8
A9	D10
A10	D12
B1	$\overline{\text{DTACK}}$
B2	$\overline{\text{LDS}}$
B3	$\overline{\text{UDS}}$
B4	D0
B5	D3
B6	D6
B7	D9
B8	D11
B9	D13
B10	D15
C1	$\overline{\text{BGACK}}$
C2	$\overline{\text{BG}}$
C3	R/$\overline{\text{W}}$
C8	D13
C9	A23
C10	A22
D1	$\overline{\text{BR}}$
D2	V_{CC}
D9	V_{SS}
D10	A21
E1	CLK
E2	V_{SS}
E9	V_{CC}
E10	A20

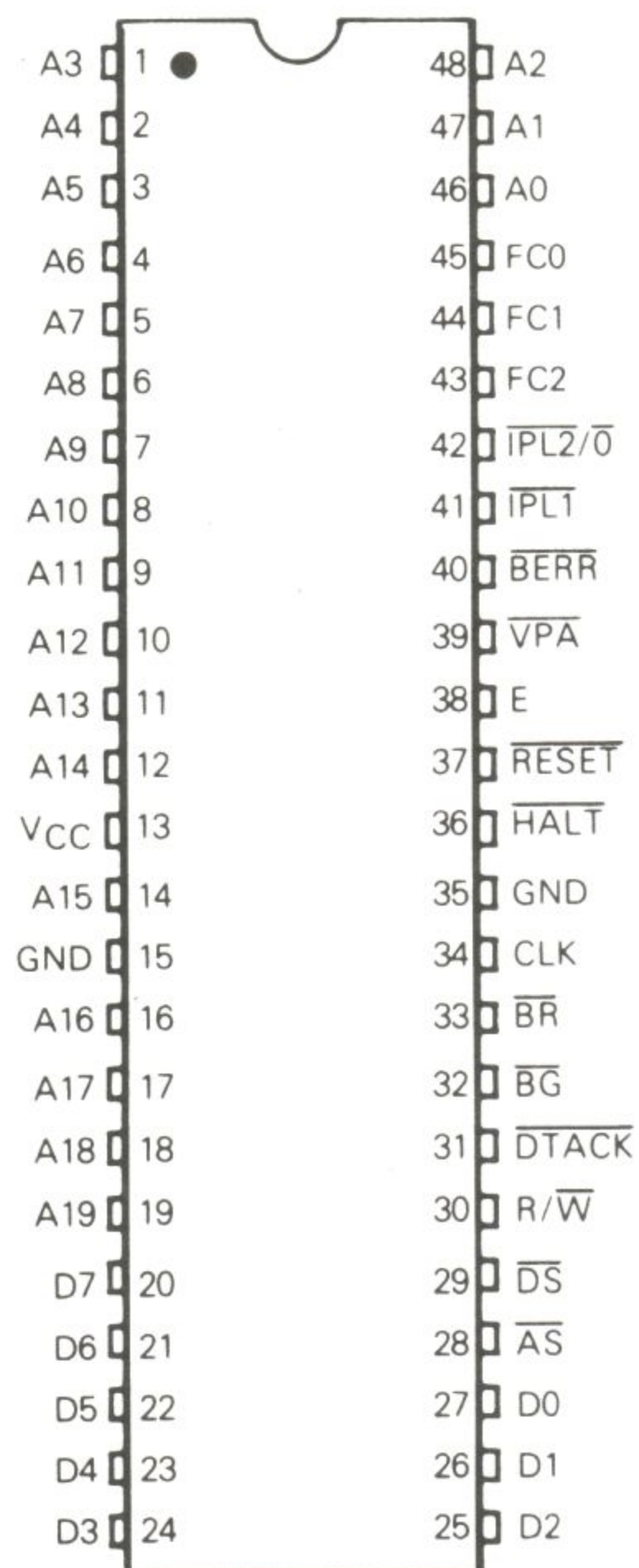
Pin Number	Function
F1	$\overline{\text{HALT}}$
F2	$\overline{\text{RESET}}$
F9	A18
F10	A19
G1	$\overline{\text{VMA}}$
G2	$\overline{\text{VPA}}$
G9	A15
G10	A17
H1	E
H2	$\overline{\text{IPL2}}$
H3	$\overline{\text{IPL1}}$
H8	A13
H9	A12
H10	A16
J1	$\overline{\text{BERR}}$
J2	$\overline{\text{IPL0}}$
J3	FC1
J4	接続しない
J5	A2
J6	A5
J7	A8
J8	A10
J9	A11
J10	A14
K1	接続しない
K2	FC2
K3	FC0
K4	A1
K5	A3
K6	A4
K7	A6
K8	A7
K9	A9
K10	接続しない

114ピン・グリッド・アレイのピン配置図（68020）

Pin Number	Function
A1	$\overline{BGACK}$
A2	A1
A3	A31
A4	A28
A5	A26
A6	A23
A7	A22
A8	A19
A9	V_{CC}
A10	GND
A11	A14
A12	A11
A13	A8
B1	N.C.
B2	$\overline{BG}$
B3	$\overline{BR}$
B4	A30
B5	A27
B6	A24
B7	A20
B8	A18
B9	GND
B10	A15
B11	A13
B12	A10
B13	A6
C1	$\overline{RESET}$
C2	CLOCK
C3	N.C.
C4	A0
C5	A29
C6	A25
C7	A21
C8	A17
C9	A16
C10	A12
C11	A9
C12	A7
C13	A5

Pin Number	Function
D1	V_{CC}
D2	V_{CC}
D3	N.C.
D4-D11	—
D12	A4
D13	A3
E1	FC0
E2	$\overline{RMC}$
E3	V_{CC}
E4-E11	—
E12	A2
E13	$\overline{OCS}$
F1	SIZ0
F2	FC2
F3	FC1
F4-F11	—
F12	N.C.
F13	$\overline{IPEND}$
G1	$\overline{ECS}$
G2	SIZ1
G3	$\overline{DBEN}$
G4-G10	—
G11	V_{CC}
G12	GND
G13	V_{CC}
H1	$\overline{CDIS}$
H2	$\overline{AVEC}$
H3	$\overline{DSACK0}$
H4-H11	—
H12	IPL2
H13	GND
J1	$\overline{DSACK1}$
J2	$\overline{BERR}$
J3	GND
J4-J11	—
J12	$\overline{IPL0}$
J13	$\overline{IPL1}$

Pin Number	Function
K1	GND
K2	$\overline{HALT}$
K3	N.C.
K4-K11	—
K12	D1
K13	D0
L1	$\overline{AS}$
L2	R/$\overline{W}$
L3	D30
L4	D27
L5	D23
L6	D19
L7	GND
L8	D15
L9	D11
L10	D7
L11	N.C.
L12	D3
L13	D2
M1	$\overline{DS}$
M2	D29
M3	D26
M4	D24
M5	D21
M6	D18
M7	D16
M8	V_{CC}
M9	D13
M10	D10
M11	D6
M12	D5
M13	D4
N1	D31
N2	D28
N3	D25
N4	D22
N5	D20
N6	D17
N7	GND
N8	V_{CC}
N9	D14
N10	D12
N11	D9
N12	D8
N13	N.C.

48ピン・デュアル・インライン・パッケージ（68008）

64ピン・デュアル・インライン・パッケージ（68000，68010）

信号	ピン	ピン	信号
D4	1	64	D5
D3	2	63	D6
D2	3	62	D7
D1	4	61	D8
D0	5	60	D9
AS	6	59	D10
UDS	7	58	D11
LDS	8	57	D12
R/W	9	56	D13
DTACK	10	55	D14
BG	11	54	D15
BGACK	12	53	GND
BR	13	52	A23
VCC	14	51	A22
CLK	15	50	A21
GND	16	49	VCC
HALT	17	48	A20
RESET	18	47	A19
VMA	19	46	A18
E	20	45	A17
VPA	21	44	A16
BERR	22	43	A15
IPL2	23	42	A14
IPL1	24	41	A13
IPL0	25	40	A12
FC2	26	39	A11
FC1	27	38	A10
FC0	28	37	A9
A1	29	36	A8
A2	30	35	A7
A3	31	34	A6
A4	32	33	A5

（68012のパッケージについては非公開）

付録E

68020の キャッシュにおける動作

プロセッサの実行時間の大部分はループを実行するのに使われる．したがって，プロセッサの効率を改善する方法として，高速メモリを**キャッシュ**とし，命令をプロセッサで容易に利用できる形のループにすることが多い．これは命令をキャッシュ・メモリに読み込み，外部メモリから命令をフェッチするよりも速く実行することが可能である．*1

「まったくウェイトのない状態」のメモリをキャッシュとして使う構成が多い．リード・サイクルの間のウェイト状態を省くと，プロセッサの実行速度は確かによくなる．68020では，「オンチップ」命令キャッシュを具体化して，この方向に進んだ．このキャッシュのアクセスはどんな外部バスでもフェッチせずに行える．

この**オンチップ・キャッシュ**は，メモリから64個のロングワード・エントリを格納でき，各エントリは「タグ」・フィールド，「有効」ビット，32ビットの命令データからできている．タグ・フィールドは上位24ビットのアドレス（A8-A31）とFC2の値を含んでいる*2（ユーザ空間とスーパバイザ空間を区別する）．

68020で命令のフェッチを要するときは，この命令がキャッシュにあるかどうかを見るために，キャッシュを最初に調べる．これは命令アドレスのA2-A7ビット*3をインデックスとしてキャッシュを調べることで行う．さらにこ

＊1　パイプライン処理を用いているので，命令のフェッチが実行に間に合わない．

＊2　4ギガバイトのメモリ空間を256バイト/ブロックで区切る．

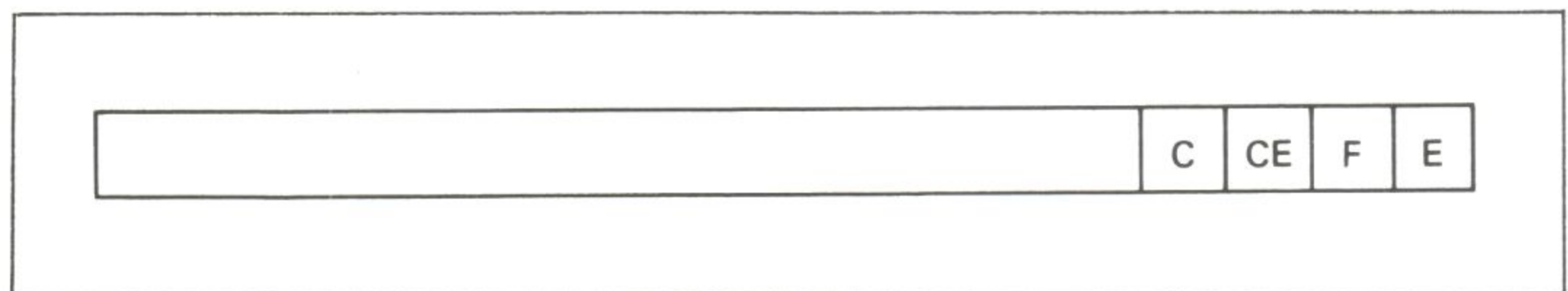
＊3　タグと有効ビットを組み合わせたものが64個あり，この6ビットでその1つを選択する．その意味でインデックスである．

のエントリに対応するタグ・フィールドとFC 2コードを調べ，それがA 8-A 31ビットと一致しており，また有効ビットがセットされていれば，キャッシュの「**ヒット**」が起こる．A 1ビットをロングワード・エントリのオフセットとしてワードを選び，プロセッサは外部フェッチをやらずに命令の実行を始める．

タグ・フィールドが一致しないか有効ビットがリセットされていると，キャッシュ・「ミス」が起こり，プロセッサは普通のリード・サイクルによって，メモリから命令をフェッチする．命令を読み込んだとき，それをキャッシュに書き込み，タグ・フィールドを更新し，有効ビットをセットする．

68020では命令のキャッシュだけが使われており，データのフェッチは外部メモリ・サイクルを常に要することに注意されたい.*

キャッシュ制御

命令キャッシュは常に必要ではなくまた，有効とも限らない．たとえばハードウェアにおけるエミュレーションでは，そのようすによってエミュレーションを容易にするため，外部フェッチを要することもある．68020では**キャッシュ制御レジスタ**（**CACR**）と**キャッシュ・アドレス・レジスタ**（**CAAR**）によって，キャッシュのアクセスを制御することができる．CAARはキャッシュ・エントリのクリアだけに使われる（下のCE参照）．

図E.1　キャッシュ制御レジスタ（CACR）

CACRは32ビット・レジスタで，**図E. 1**に示すように，4ビットだけが使われる．そのアクセスはMOVEC（制御の転送；move control）命令で行うが，これは特権命令である．CACRのこの4ビットは次のようになっている．

E——キャッシュの使用可能

このビットがセットされていると，普通のキャッシュ機能が使用可能になる．クリアされていると，キャッシュの使用が禁止され，命令のフェッチを外部サ

---

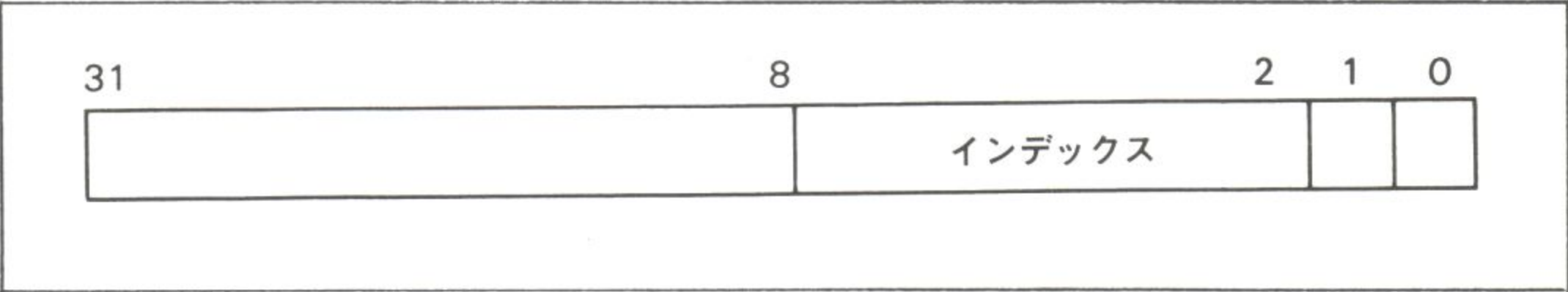
* 命令はキャッシュにあるので，命令の先読み（プリフェッチ）とデータの読込みが並行して行える．

イクルで行う．ハードウェアをリセットすると，このビットは常にクリアされる．

F——キャッシュの凍結

このビットがセットされていると，キャッシュの内容がロックされ，更新できなくなる．[*1] それでもキャッシュ・ヒットは起こり得るが，キャッシュ・ミスによるキャッシュ・エントリの置換えはできない．ハードウェアをリセットすると，このビットも常にクリアされる．

CE——エントリのクリア

このビットがセットされていると，CAARで与えられたインデックスに対応するキャッシュ・エントリの有効ビットがクリアされる．[*2] この動作はCACRをMOVEC命令で設定したときだけ使える（ビットは書出し専用である）．**図E.2**にCAARのフォーマットを示した．

図E.2 キャッシュ・アドレス・レジスタ(CAAR)

C——キャッシュのクリア

この書出し専用ビットをMOVEC命令でセットすると，キャッシュ・エントリはすべて無効になる（たとえばタスク・コンテキスト・スイッチの一部）．[*3]

キャッシュの使用禁止

キャッシュ・ディスエーブル入力（$\overline{\text{CDIS}}$）信号は，キャッシュを動的に使用禁止にする．外部ロジックが$\overline{\text{CDIS}}$をアサートした後で，次のどんな命令のフェッチも外部メモリを経由するようになり，CACRのキャッシュ・イネーブル・ビットEの状態に関係しない．外部ロジックが$\overline{\text{CDIS}}$をネゲートすると，次の命令のフェッチから再び使用可能になる．

＊1　よく使われるルーチンがキャッシュにあるときは有効である．

＊2　タグには関係しない．またこれによって，命令キャッシュへの入力を禁止する．

＊3　マルチタスク・システムで，タスクを切り換えるためのスイッチ．この場合，前のタスクの状態を保存する必要がある．

日本語版補遺

本補遺は，本文を補足説明するために，日本語版につけ加えたものである．

補遺Ⅰでは，68000系アセンブラ語で，アセンブラ・プログラムを作成する場合の，基本的知識——文法について説明する．また補遺Ⅱは，機械語命令の中のコプロセッサ命令について補足するものである．

日立プロセスコンピュータエンジニアリング（株）
技術教育センタ長　加藤木和夫

Ⅰ．アセンブラの文法

1．書式
2．ラベル
3．命令実行文の書き方
4．アドレッシング・モードの書き方
5．アセンブラ制御文
6．データ形式
7．例

Ⅱ．コプロセッサの命令について

1．コプロセッサ命令
2．MC 68881で定義される命令

I

アセンブラの文法

1. 書　　式

アセンブラ・プログラムは複数の文からなり，文には次の2種類がある．

- 実行命令文
- アセンブラ制御文

実行命令文は，第4章に記述した機械語命令に1対1で対応している．一方，アセンブラ制御文は，機械語命令に対応する文ではなく，アセンブラ・プログラムを機械語に変換する処理プログラム（すなわちアセンブラ）への指定文である．

ラベル・フィールド	区切り	オペレーション・フィールド	区切り	オペランド・フィールド	区切り	コメント・フィールド	改行

付図1　文の書式

ラベル・フィールド	オペレーション・フィールド	オペランド・フィールド	コメント・フィールド
	MOVE. L	D1, D3	
TOP	ADD. W	D2, D3	
	SUBQ. W	♯100, D3	
	BNE	TOP	; LOOP
	RTS		

付図2　文の書式例

アセンブラ・プログラムは，この2種類の文を組み合わせて作成するわけであるが，文の書き方には一定の書式がある．

付図1に文の書式を示す．

アセンブラ・プログラムは1行1文が基本で，1文が複数行にまたがることはない．しかしアセンブラによっては，1行に複数個の文を書くことができる．

付図1において，各フィールドは必要がなければ書かなくともよい．改行だけからなる文——空文——は無視されるのが普通である．

各フィールドの間には，区切りとして，1文字以上の空白が必要である．アセンブラによっては，コメント・フィールドの始めに特殊文字——たとえばセミコロン(;)——を書くこともあるが，詳細は各アセンブラ言語マニュアルを参照されたい．

付図2に一例を示す．

2. ラベル

2.1　ラベルの意味と書き方

ラベルは，プログラムやデータの位置（ロケーション）を示すものである．たとえば，分岐命令（BRA，BEQ，……）の飛び先，データを格納するメモリ領域，記号名の定義などに用いる．

●例1　分岐命令の飛び先

●例2　データの格納場所

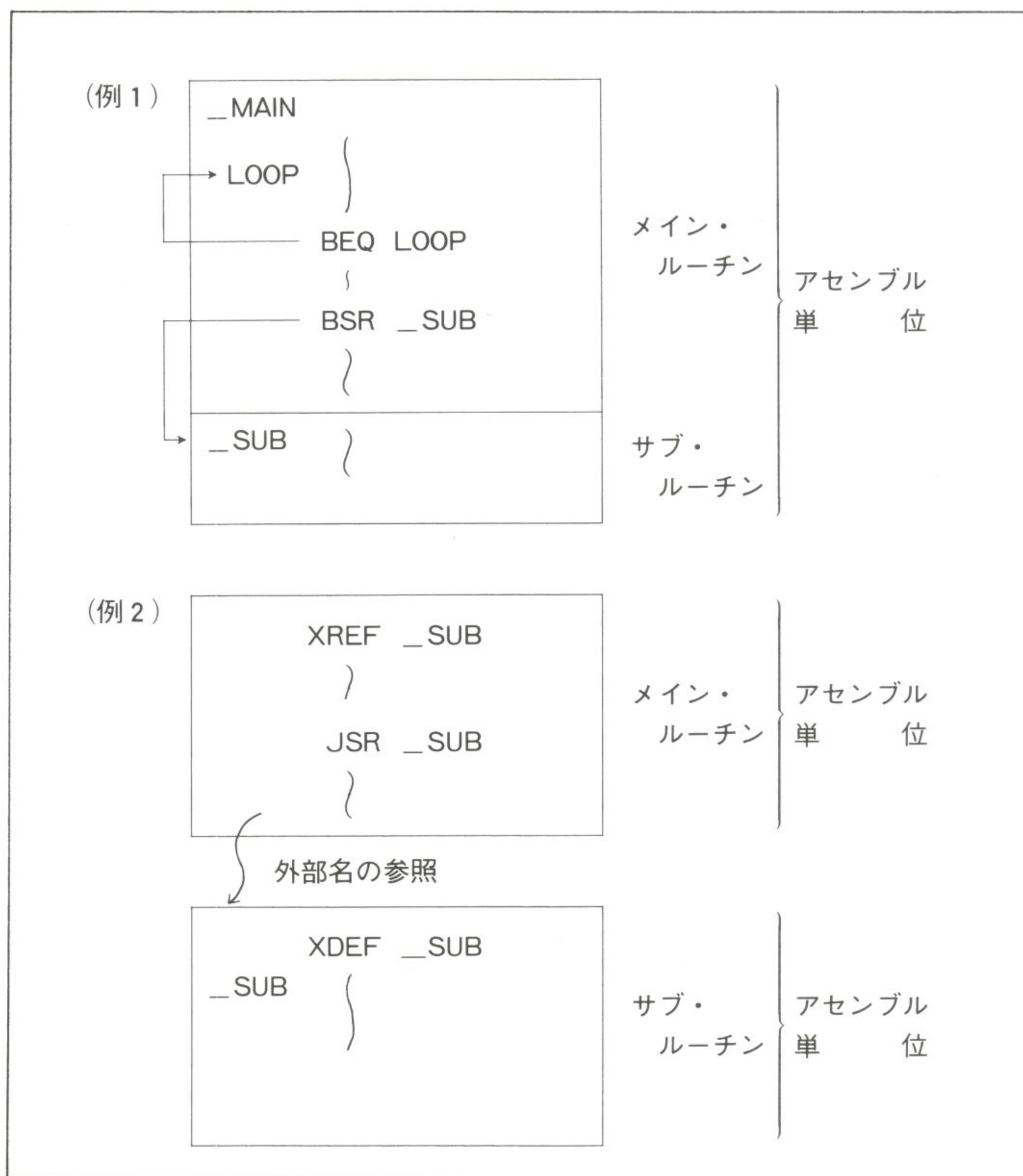
```
COUNT    DS    1
```

●例3　記号名の定義

```
CASE    EQU    10
```

ラベルは英数字の列で表現する．文字の個数はアセンブラによって異なり，また英数字以外の特殊文字（たとえばアンダスコア）を何字か使えるアセンブラも多い．

なお，先頭の文字が数字であってはならない．

（注）ラベルと次のオペランド・フィールドの区切り記号として，コロン(:)を用いるアセンブラもある．

付図3　ラベルの有効範囲

2.2 ラベルの有効範囲

プログラムやデータにおいて用いたラベルは，アセンブル単位内でしか参照できない．このことを，ラベルには有効範囲があるという．すなわち，ラベルの有効範囲は，このラベルを含むプログラムを単位として定められている．

ただし，外部からラベルを参照できる方法もある．これにはアセンブラ制御文を使用する．**付図3**に例を示す．

●例1　アセンブル単位内（内部名）

LOOP，__SUB がラベルである．__MAIN や__SUB のように，ルーチン名の先頭にアンダスコアをつけると識別しやすい．

●例2　外部から参照できるラベル（外部名）

アセンブラ制御文 XDEF に現われたラベルは，別のアセンブル単位から参照できる．ここで，外部参照を示すアセンブラ制御文 XDEF，XREF はアセンブラによって表現が異なるので，注意されたい．

3. 命令実行文の書き方

命令実行文は機械語命令と1対1に対応する文であり，その記述は，オペレーション・フィールドとオペランド・フィールドに分けて書く．

オペレーション・フィールドには機械語命令のニーモニック・コードを記述し，オペランド・フィールドにはオペランド（操作するデータ）を記述する．

オペレーション・フィールドのニーモニック・コードに続いて，操作するデータの大きさを指定する「オペランド・サイズ」を記述する．

オペランド・サイズの指定は，B（バイト），W（ワード），L（ロングワード）の3種である．書き方の一例を示す．

```
MOVE.L    D1, D2
```

なお，機械命令によっては暗黙にオペランド・サイズが定まっているものもあり，その場合は記述する必要がない．また，実数データを扱う場合はオペランド・サイズの種類が増えるが，これについては**補遺Ⅱ.2**を参照されたい．

オペレーション・フィールドの次のオペランド・フィールドには，処理すべきデータまたはそのアドレスを書く．命令実行文の多くはデータを処理するもので，68K の場合はオペランドを2個記述する形をとり，オペランドとオペランドの間はコンマ(,)で区切る．

付表1（142～146 ページ）に，機械語命令のニーモニックとオペランド・サイズ，オペランド・シンタックスおよびその具体例を示す．

付表1　実行命令文

分類	機械語命令のニーモニック	オペランド・サイズ	オペランドのシンタックス	使用例（〈ea〉は付表2参照）
データ転送命令	EXG	L	Rn, Rn	EXG　D1, D2
	LEA	L	〈ea〉, An	LEA　〈ea〉, A1
	LINK	W, L	An, ＃〈ディスプレースメント〉	LINK　A1, ＃−18
	MOVE	B, W, L	〈ea〉, 〈ea〉	MOVE.L　〈ea〉, 〈ea〉
	MOVEA	W, L	〈ea〉, An	MOVEA.L　〈ea〉, A1
	MOVEC	L	Rn, RcまたはRc, Rn	MOVEC　USP, A1
	MOVEM	W, L	リスト, 〈ea〉	MOVEM.L　D1−D7／A1−A6, −(A7)
			〈ea〉, リスト	MOVEM.L　(A7)＋, D1−D7／A1−A6
	MOVEP	W, L	Dn, (d_{16}, An)	MOVEP.L　D1, (10, A1)
			(d_{16}, An), Dn	MOVEP.L　(10, A1), D1
	MOVEQ	L	＃〈データ〉, Dn	MOVEQ　＃1, D1
	PEA	L	〈ea〉	PEA　〈ea〉
	UNLK	L	An	UNLK　A1
整数算術演算命令	ADD	B, W, L	Dn, 〈ea〉	ADD.L　D1, 〈ea〉
			〈ea〉, Dn	ADD.L　〈ea〉, D1
	ADDA	W, L	〈ea〉, An	ADDA.L　〈ea〉, A1
	ADDI	B, W, L	＃〈データ〉, 〈ea〉	ADDI.L　＃$FFFFFF, 〈ea〉
	ADDQ	B, W, L	＃〈データ〉, 〈ea〉	ADDQ.L　＃1, 〈ea〉
	ADDX	B, W, L	Dn, Dn	ADDX.L　D1, D2
			−(An), −(An)	ADDX.L　−(A1), −(A2)
	CLR	B, W, L	〈ea〉	CLR.W　〈ea〉
	CMP	B, W, L	〈ea〉, Dn	CMP.W　〈ea〉, D1
	CMPA	W, L	〈ea〉, An	CMPA.L　〈ea〉, A1
	CMPI	B, W, L	＃〈データ〉, 〈ea〉	CMPI.L　＃1, 〈ea〉
	CMPM	B, W, L	(An)＋, (An)＋	CMPM.L　(A1)＋, (A2)＋
	CMP2	B, W, L	〈ea〉, Rn	CMP2.W　〈ea〉, D1
	DIVS	W	〈ea〉, Dn	DIVS.W　〈ea〉, D1
		L	〈ea〉, Dq	DIVS.L　〈ea〉, D1
		L	〈ea〉, Dr：Dq	DIVS.L　〈ea〉, D1：D2
	DIVU	W, L	DIVSに同じ	
	DIVSL	L	〈ea〉, Dr：Dq	DIVSL.L　〈ea〉, D1：D2
	DIVUL	L	〈ea〉, Dr：Dq	DIVUL.L　〈ea〉, D1：D2
	EXT	W, L	Dn	EXT.W　D1
	EXTB	L	Dn	EXTB.L　D1
	MULS	W	〈ea〉, Dn	MULS.W　〈ea〉, D2
		L	〈ea〉, Dl	MULS.L　〈ea〉, D2
		L	〈ea〉, Dh：Dl	MULS.L　〈ea〉, D1：D2
	MULU	W, L	MULSに同じ	
	NEG	B, W, L	〈ea〉	NEG.L　〈ea〉
	NEGX	B, W, L	〈ea〉	NEGX.L　〈ea〉
	SUB	B, W, L	〈ea〉, Dn	SUB.L　〈ea〉, D1
		B, W, L	Dn, 〈ea〉	SUB.L　D1, 〈ea〉

分類	機械語命令のニーモニック	オペランド・サイズ	オペランドのシンタックス	使用例（〈ea〉は付表2参照）
整数算術演算命令	SUBA	W，L	〈ea〉，An	SUBA.L　〈ea〉，A1
	SUBI	B，W，L	＃〈データ〉，〈ea〉	SUBI.W　＃100，〈ea〉
	SUBQ	B，W，L	＃〈データ〉，〈ea〉	SUBQ.L　＃1，〈ea〉
	SUBX	B，W，L	Dn，Dn	SUBX.L　D1，D2
			−(An)，−(An)	SUBX.L　−(A1)，−(A2)
論理命令	AND	B，W，L	〈ea〉，Dn	AND.L　〈ea〉，D1
			Dn，〈ea〉	AND.L　D1，〈ea〉
	ANDI	B，W，L	＃〈データ〉，〈ea〉	ANDI.L　＃1，D1
	EOR	B，W，L	Dn，〈ea〉	EOR.L　D1，〈ea〉
	EORI	B，W，L	＃〈データ〉，〈ea〉	EORI.L　＃$FFFFFFFF，〈ea〉
	NOT	B，W，L	〈ea〉	NOT.L　〈ea〉
	OR	B，W，L	〈ea〉，Dn	OR.L　〈ea〉，D1
			Dn，〈ea〉	OR.L　D1，〈ea〉
	ORI	B，W，L	＃〈データ〉，〈ea〉	ORI.L　＃1，〈ea〉
	Scc	B	〈ea〉	SEQ　〈ea〉
	TST	B，W，L	〈ea〉	TST.B　〈ea〉
シフトおよびローテイト命令	ASL	B，W，L	Dn，Dn	ASL.L　D1，D2
		B，W，L	＃〈データ〉，Dn	ASL.L　＃2，D1
		W	〈ea〉	ASL.W　〈ea〉
	ASR LSL LSR ROL ROR ROXL ROXR	ASLに同じ		
	SWAP	L	Dn	SWAP　D1
ビット命令	BCHG	B，L	Dn，〈ea〉	BCHG.L　D1，〈ea〉
			＃〈データ〉，〈ea〉	BCHG.L　＃1，〈ea〉
	BCLR BSET BTST	BCHGに同じ		
ビットフィールド命令	BFCHG	—	〈ea〉｛off：w｝	BFCHG　〈ea〉｛8：2｝
	BFCLR	—	〈ea〉｛off：w｝	
	BFEXTS	—	〈ea〉｛off：w｝，Dn	BFEXTS　〈ea〉｛8：2｝，D1
	BFEXTU	—	〈ea〉｛off：w｝，Dn	
	BFFFO	—	〈ea〉｛off：w｝，Dn	
	BFINS	—	Dn，〈ea〉｛off：w｝	BFINS　D1，〈ea〉｛8：2｝
	BFSET	—	〈ea〉｛off：w｝	BFSET　〈ea〉｛8：2｝
	BFTST	—	〈ea〉｛off：w｝	

分類	機械語命令のニーモニック	オペランド・サイズ	オペランドのシンタックス	使用例（〈ea〉は付表2参照）
2進化10進命令	ABCD	B	Dn, Dn	ABCD　D1, D2
			−(An), −(An)	ABCD　−(A1), −(A2)
	NBCD	B	〈ea〉	NBCD　〈ea〉
	PACK	—	−(An), −(An), #〈データ〉	PACK　−(A1), −(A2), #1
			Dn, Dn, #〈データ〉	PACK　D1, D2, #1
	SBCD	B	Dn, Dn	SBCD　D1, D2
			−(An), −(An)	SBCD　−(A1), −(A2)
	UNPK	—	−(An), −(An), #〈データ〉	UNPK　−(A1), −(A2), #1
			Dn, Dn, #〈データ〉	UNPK　D1, D2, #1
プログラム制御命令	Bcc	B, W, L	〈ラベル〉	BEQ.S　LOOP
	BRA	B, W, L	〈ラベル〉	BRA.S　ERA
	BSR	B, W, L	〈ラベル〉	BSR.S　MAX
	CALLM	—	#〈データ〉, 〈ea〉	CALLM　#4, 〈ea〉
	DBcc	W	Dn, 〈ラベル〉	DBLE　D1, LAB
	JMP	—	〈ea〉	JMP　〈ea〉
	JSR	—	〈ea〉	JSR　〈ea〉
	NOP	—	—	NOP
	RTD	W	#〈ディスプレースメント〉	RTD　#8
	RTE	—	—	RTE
	RTM	—	Rn	RTM　A7
	RTR	—	—	RTR
	RTS	—	—	RTS
システム制御命令	ANDI	W	#〈データ〉, SR	ANDI.W　#$FFFF, SR
		B	#〈データ〉, CCR	ANDI.B　#0, CCR
	BKPT	—	#〈データ〉	BKPT　#1
	CHK	W, L	〈ea〉, Dn	CHK.L　〈ea〉, D1
	CHK2	B, W, L	〈ea〉, Rn	CHK2.L　〈ea〉, A1
	EORI	W	#〈データ〉, SR	EORI.W　#1, SR
		B	#〈データ〉, CCR	EORI.B　#1, CCR
	ILLEGAL	—	—	ILLEGAL
	MOVE	W	〈ea〉, SR	MOVE.W　〈ea〉, SR
		W	SR, 〈ea〉	MOVE.W　SR, 〈ea〉
		W	〈ea〉, CCR	MOVE.W　〈ea〉, CCR
		W	CCR, 〈ea〉	MOVE.W　CCR, 〈ea〉
		L	USP, An	MOVE.L　USP, A1
		L	An, USP	MOVE.L　A1, USP
	MOVEC	L	Rc, Rn	MOVEC.L　USP, D1
		L	Rn, Rc	MOVEC.L　D1, USP
	MOVES	B, W, L	Rn, 〈ea〉	MOVES.L　D1, 〈ea〉
			〈ea〉, Rn	MOVES.L　〈ea〉, D1

分類	機械語命令のニーモニック	オペランド・サイズ	オペランドのシンタックス	使用例（〈ea〉は付表2参照）
システム制御命令	RESET	—	—	RESET
	STOP	W	#〈データ〉	STOP #1
	TRAP	—	#〈データ〉	TRAP #2
	TRAPcc	—	—	TRAPEQ
		W, L	#〈データ〉	TRAPEQ.L #1
	TRAPV	—	—	TRAPV
マルチプロセッサ命令	CAS	B, W, L	Dc, Du, 〈ea〉	CAS.W D1, D2, 〈ea〉
	CAS2	W, L	Dc1：Dc2, Du1：Du2, (Rn)：(Rn)	CAS2.W D1：D2, D2：D2, (A1)：(A2)
	cpBcc	W, L	〈ラベル〉	補遺II参照のこと．
	cpDBcc	W	〈ラベル〉, Dn	
	cpGEN	ユーザ定義		
	cpRESTORE	—	〈ea〉	
	cpSAVE	—	〈ea〉	
	cpScc	B	〈ea〉	
	cpTRAPcc	—	—	
		W, L	#〈データ〉	
	TAS	B	〈ea〉	TAS.B 〈ea〉

An　アドレス・レジスタ
Dn　データ・レジスタ
Rn　アドレスまたはデータ・レジスタ
Rc　制御レジスタ（USPなど）
〈ea〉　実効アドレス
d_{16}　16ビットのディスプレースメント

#〈データ〉　イミディエイト・データ
リスト　レジスタのリスト（たとえば　D3-D5）
{off：w}　ビット・フィールド（offはオフセット，wはビット幅）

Dr　データ・レジスタ（除算の余り）
Dq　データ・レジスタ（除算の商）
Dc　データ・レジスタ（比較に使用）
Du　データ・レジスタ（更新に使用）
Dh　データ・レジスタ（乗算した結果の上位32ビット）
Dl　データ・レジスタ（乗算した結果の下位32ビット）

SR　アクティブ・スタック・ポインタ
CCR　コンディション・コード・レジスタ
USP　ユーザ・スタック・ポインタ

cc　コンディション・コードを意味し，次の種類がある（Bcc，cpBcc）.

CC	キャリー・クリア	LS	ロウまたは同じ
CS	キャリー・セット	LT	より小さい
EQ	等しい	MI	負（minus）
GE	大きいまたは等しい	NE	等しくない
GT	より大きい	PL	正（plus）
HI	ハイ	VC	オーバフロー・クリア
LE	小さいまたは等しい	VS	オーバフロー・セット

付表1のオペランド・シンタックスに現われる〈ea〉は，effective address の略で実効アドレスを意味する．これはオペランドすなわち処理すべきデータが格納されている場所を示すアドレスのことである．実効アドレスの表し方にはさまざまな方法がある．たとえば，オペランドがレジスタ上にあれば，オペランド・フィールドにレジスタ名を書く．すなわちデータ・レジスタ3番ならばD3になる．一方，オペランドがメモリ上にあれば，レジスタ間接やメモリ間接などの指定方法がある．このように，実効アドレスを指定する方法がアドレッシング・モードである．その詳細については，第3章および次節を参照されたい．

（注）オペランド・サイズ

分岐命令（Bcc，BRA，BSR，DBcc）のオペランド・サイズでは，バイト・オフセットを指定するのに .Sを使い，指定がなければワード・オフセットとするアセンブラが多い．

分岐命令はデータを処理する文ではないので，オペランド・サイズを指定するB，W，Lの意味が多少異なるためであろう．Sはshort の略と思われる．

●例

4. アドレッシング・モードの書き方

アドレッシング・モードの種類とそれに対応するアセンブラ・シンタックスを**付表2**に示す．

アセンブラ・シンタックスはシステムにより少し異なるため，コーディングするときには各システムのアセンブラ・マニュアルを参照されたい．また，機械語命令によっては使用できるアドレッシング・モードに制限があるので，それらについても各マニュアルを参照されたい．

付表2　アドレッシング・モード

アドレッシング・モード		アセンブラ・シンタックス	例
レジスタ直接	データ・レジスタ直接	Dn	D1
	アドレス・レジスタ直接	An	A1
レジスタ間接	アドレス・レジスタ間接	(An)	(A1)
	ポストインクリメントつき アドレス・レジスタ間接	(An)+	(A1)+
	プリデクリメントつき アドレス・レジスタ間接	−(An)	−(A1)
	〈ディスプ〉つきアドレス・レジスタ間接	(d_{16}, An)	(500, A1)
	インデックス・8ビット〈ディスプ〉つき アドレス・レジスタ間接	(d_8, An, Xn..サイズ)	(16, A1, D2.W)
	ベース〈ディスプ〉つき アドレス・レジスタ間接	(d_8, An, Xn.サイズ*スケール) (bd, An, Xn.サイズ*スケール)	(16, A1, D2.L*4)
メモリ間接	ポストインデックスつきメモリ間接	([bd, An], Xn.サイズ*スケール, od)	([16, A1], D1.L*4, 100)
	プリインデックスつきメモリ間接	([bd, An], Xn.サイズ*スケール], od)	([16, A1, D1.L*4], 100)
プログラム・カウンタ(PC)間接	〈ディスプ〉つきPC間接	(d_{16}, PC)	(16, PC)
	インデックス〈ディスプ〉つきPC間接	(dn, PC, Xn.サイズ)	(16, PC, D2.L)
		(dn, PC, Xn.サイズ*スケール)	(16, PC, D2.L*4)
PCメモリ間接	ポストインデックスつき PCメモリ間接	([bd, PC], Xn.サイズ*スケール, od)	([10, PC], D1.L*4, 100)
	プリインデックスつき PCメモリ間接	([bd, PC], Xn.サイズ*スケール], od)	([10, PC, D1.L*4], 100)
絶対アドレス	絶対ショートアドレス	xxxx.W	$2000.W(16進)
	絶対ロングアドレス	xxxx.L	$20000.L(16進)
イミディエイト・データ		#xxxx.サイズ	#10.B

Dn：データ・レジスタn
　(たとえば，D3はデータ・レジスタ3)
An：アドレス・レジスタn
　(たとえば，A3はアドレス・レジスタ3)
Xn：インデックス・レジスタn
　(データまたはアドレス・レジスタn)
PC：プログラム・カウンタ
〈ディスプ〉：ディスプレースメント
d_8：8ビットのディスプレースメント
d_{16}：16ビットのディスプレースメント
bd：ベース・ディスプレースメント
od：外部ディスプレースメント (outer displacement)
サイズ：インデックス・レジスタのサイズ選択子で，
　Wを指定すると符号拡張ワード，
　Lを指定すると符号つきロングワード
スケール：インデックス・レジスタのスケール・ファクタで
　1，2，4，8を指定する．
xxxx：値

5. アセンブラ制御文

アセンブラ制御文は，アセンブラに対して指示を与える文で，機械語命令に対応する文ではない．ただし，データ定義とメモリ領域を確保する文は機械語命令に対応していないが，実際にメモリ領域を確保し，データ値を設定する．

アセンブラ制御文の種類および書き方は，アセンブラによってかなり異なるので，各システムのアセンブラ・マニュアルを参照されたい．

以下には，主要なものの一例を示す．（　）内は一般によく使われる識別名の例である．

(1)　アセンブリ制御に関する文

- セクションの指定（SECTION）
- 絶対アドレスの指定（ORG）
- プログラム終了の指示（END）

(2)　記号名の値を指定する文

（EQU など）

(3)　データ定義およびメモリ領域確保のための文

- 定数データの定義（DC）
- メモリ領域の確保（DS）

(4)　外部名に関する文

アセンブル単位にまたがって名前が参照できるようにする文など（XREF, XDEF）

(5)　アセンブル時の条件を設定する文

(6)　リスティング・フォーマットを指定する文

6. データ形式

アセンブラにおけるデータ形式について，定数，記号名，式に分けて次に説明する．

(1)　定数

定数には数値定数と文字型定数があり，数値定数は整数値定数と実数値定数に分かれる．以下にそれぞれの表記法を示す．

a.　整数値定数（整数）

整数値定数は16進数または10進数で表す．特に断わらなければ普通は，10進数である．16進数は，頭に＄をつけるのが一般的である．このほか，8進数や2進数の使えるアセンブラもある．

●例　　256　　(10進数)
　　　　$ 2000　(16進数)

b. 実数値定数（実数）

実数値定数は10.8のように小数点を含む数を指す．その表記法は，アセンブラによって異なる（あるいは実数そのものが使えない）ため，詳細は各システムのアセンブラ・マニュアルを参照されたい．

c. 文字型定数

これは，ASCIIまたはJISコードを文字で表すのに使用する．普通はアポストロフィ(')で文字の前後を囲むが，アセンブラによってはダブルクォーテーション(")で囲む．

●例

'JAPAN', "JAPAN"

(2) 記号名

ラベルには記号名を用いる．記号名は，英字（A～Z，a～z）と数字（0～9）の列で，英字から始まる．これら以外にアンダスコア（_）を使えるアセンブラが多い．

●例　　LOOP
　　　　_MAIN

なお3AHなどは，数字で始まっているので，記号名ではない．

(3) 式

普通のアセンブラでは，文のオペランドに式を記述できる．記述できる個所は，命令実行文のイミディエイト・データやディスプレースメント，そしてアセンブラ制御文の多くのオペランドである．ただし，式は単純なものである．

式を使うと，プログラムの意味がわかりやすくなったり，変更しやすくなる．

たとえば，メモリ領域を確保するとき

```
AREA    DS  50
```

と書くより

```
AREA    DS  5*10
```

とする方が，50の意味するところがわかりやすく，さらに記号名を導入して

```
ENTRY   EQU  5
CASE    EQU  10
AREA    DS   ENTRY*CASE
```

と書くと，なおわかりやすくなる．

式で使える演算子は，＋（加算），－（減算），＊（乗算），／（除算）などであるが，詳細はアセンブラにより異なるので，マニュアルを参照されたい．

7．例

付図 4 に簡単な使用例を示す．これはサブルーチンの例で，パラメータはすべてデータ・レジスタを使って受け渡している．

```
_COUNT   MOVE.L   D1, D3
         SUB.L    D2, D3
         BEQ      EQ
         BGT      GT
         MOVE.L   #-1, D4
         BRA      MODORI
EQ       MOVE.L   #0, D4
         BRA      MODORI
GT       MOVE.L   #1, D4
MODORI   RTS                     ; return
         END
D1とD2に値がセットされている.
結果はD4にセットされる.
```

付図 4　使用例

Ⅱ

コプロセッサ命令について

1. コプロセッサ命令

第４章の命令セットの 表4.10 マルチプロセッサ命令 に関し，補足説明を加える．

68000の汎用マイクロプロセッサ・ファミリは，付加プロセッサであるコプロセッサを組み合わせることで，機能拡張あるいは機能分散を図ることができる．汎用プロセッサに対して，コプロセッサは複数台連結できる．

システム設計者が汎用プロセッサにコプロセッサを連結する場合，一般には詳細な「コプロセッサ・インターフェース」の知識を必要とする．しかし，Motorola 社のコプロセッサを連結する場合は，詳細な知識を必要としない．

現在，Motorola 社のコプロセッサには次の２種類がある．

- MC68851　　ページ式メモリ管理ユニット（PMMU）
- MC68881　　浮動小数点演算用コプロセッサ（FPCP）

これらのコプロセッサは，Motorola 社で定義した命令を実行する．どのような命令が定義されているかは，おのおののユーザーズ・マニュアルを参照する必要がある．表4.10に記述されたコプロセッサ命令と，上に述べたコプロセッ

サで定義した命令との関係については，後述する．

プログラマから見た場合は，汎用プロセッサに備わっている命令セットにプラスして，コプロセッサの命令セットが使えるので，たんに命令セットが増えたと考えればよい．

たとえば，MC68881浮動小数点演算コプロセッサを付加したシステムのアセンブラ・マニュアルを見ると，通常の命令セットと合わせて浮動小数点演算の命令セットが記述されている．プログラマは，コプロセッサをまったく意識することなく，プログラミングすることができる．

コプロセッサは，汎用のメインプロセッサと同期をとりながら動作する．すなわち，メインプロセッサは，命令を実行中にコプロセッサ命令を認識すると，命令コードをコプロセッサ側へ渡し，結果をもらってから次の命令の実行へと進む．

詳細は，参考文献〔1〕を参照されたい．

2. MC68881で定義される命令

第4章の表4.10に記述されたコプロセッサ命令（cpGEN，cpBcc，……）を詳しく見てみよう．

コプロセッサ命令の1ワード目は，**付録A**（118ページ）に記述されているように「1111」(15〜12ビット)というオペレーション・コードで始まる．次の3ビット（11〜9ビット）は，Cp-IDすなわちコプロセッサの識別番号を表している．残りのビット（8〜0ビット）は，命令によって異なる．

ここで，cpGEN(データ処理用命令などに使用する汎用のコプロセッサ命令)と，MC68881で定義されている加算命令FADDを比較してみよう．それぞれの命令フォーマットを**付図5**に示す．この図で見るように，cpGENがMC68881で定義されるFADDなどの浮動小数点演算命令の汎用形になっていることは，容易に想像できるであろう．

プログラマはMC68881を使用するにあたって，cpGENではなくFADDなどの定義済みの命令を使えばよいということになる．

MC68881でサポートする命令のアセンブラ・シンタックスは，汎用（メイン）プロセッサの書き方に準ずる．ただし，オペランド・サイズは，B，W，Lのほかに，S，D，X，Pを指定できる．ここで，S，D，X，Pは

S：実数型（4バイト）

D：倍精度実数型（8バイト）

X：拡張実数型（12バイト）

P：Packed Decimalの実数型

である．

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	1	1	Cp－ID			0	0	0	実効アドレス					
										モード			レジスタ		
コプロセッサ用コマンド・ワード															

cpGEN

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	1	1	0	0	1	0	0	0	実効アドレス					
										モード			レジスタ		
0	R/M	0	ソース・スペシファイヤ			デスティネーション・レジスタ			0	1	0	0	0	1	0

FADD

Cp－ID：001（68881）
実効アドレス：R／M＝0のときは使われない．
R／M＝1のときはアドレッシング・モードを示す．
R／M：0のときは〈レジスタ〉→〈レジスタ〉
1のときは〈実効アドレス〉→〈レジスタ〉
ソース・スペシファイヤ：R／M＝0のときは，ソースのFPm*を示す．
R／M＝1のときは，ソースのデータ・フォーマット（B，W，L，S，D，X，P）を示す．
デスティネーション・レジスタ：デスティネーションのFPn*を示す．

＊FPm，FPnは浮動小数点データ・レジスタである．
（注）　3ワード目以降は省略．

付図5　cpGENとFADD

例

```
FADD. S  D1, FP1
FADD. S  FP1, (A1)
```

ただしFPnは浮動小数点データ・レジスタである．

各命令と指定できるオペランドの種類およびオペランド・サイズの組み合わせは，各システムのアセンブラ・マニュアルを参照されたい．

使用例を**付図6**に示す．

```
MOVE.L     #10.2, D1
MOVE.L     #5.4, D2
FMOVE.S    D1, FP1        10.2→FP1
FADD.S     D2, FP1        10.2+5.4→15.6
FINTRZ.X   FP1, FP2       15.6→15.0
FMOVE.L    FP2, D3        15.0→15→D3
FMOVE.S    FP2, D4        15.0→D4
```

付図6　浮動小数点演算の例

参考文献

本文・付録

鎌倉三郎　MC68020 の概要　bit, 1985. 1　共立出版

須山徹郎　命令キャッシュとコプロセサ・インタフェースを内蔵する 68020 日経バイト, 1985. 5　日経マグロウヒル

MC 68000 User's Manual　Motorola

16 Bit User's Manual　Motorola

岡本他　ザ 68000——ハードウェア・ソフトウェア・アプリケーション 共立出版

日本語版補遺

〔1〕"MC 68020 32-Bit Microprocessor User's Manual second Edition", Prentice-Hall
(MC 68020 ユーザーズ・マニュアル, CQ 出版社)

〔2〕"MC 68881 FLOATING-POINT COPROCESSOR USER'S MANUAL", MOTOROLA, May, 1985

索　　引

ナ　行

ハ　行

マ　行

ヤ　行

ラ　行

ワ　行

アルファベット

数　字

訳者紹介

岡本　茂（おかもと　しげる）

1960年　東京教育大学理学研究科博士課程修了

現　在　茨城大学理学部数学科教授・理学博士

主要著訳書　マイクロコンピュータ辞典（共訳編，共立出版）
ザ68000（共著，共立出版）
ソフトウェア開発環境（共著，共立出版）
UNIX 入門（著，日刊工業新聞）

68000ファミリハンドブック
——68000，68008，68010，68020——

1987年5月22日　第1刷発行

著　者　ウィリアム・クレイマー
　　　　ゲリー・ケイン

訳　者　岡　本　　茂

発行所　啓学出版株式会社
代表者　三　井　数　美

郵便番号　101
東京都千代田区神田神保町1-46
電　話　東京03(233)3731(代)
振　替　東京3－109286

印刷・製本／株式会社　廣　済　堂
ISBN 4-7665-0521-2
本書の定価はカバーに表示してあります

Printed in Japan
N. S／松岡

68000ファミリ ハンドブック
W·クレイマー＋G·ケイン·著
岡本　茂·訳

「出」1953
啓学出版
定価2200円

68020

ISBN4-7665-0521-2 C3055 ¥2200E

68000ファミリ ハンドブック 68000·68008·68010·68020 W·クレイマー＋G·ケイン·著 岡本 茂·訳

啓学出版